

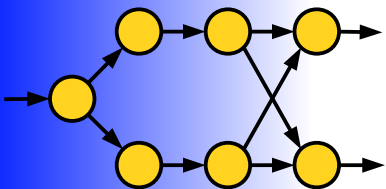
On Runtime Systems for Task-based Programming on Heterogeneous Platforms

Samuel Thibault
SATANAS/STORM

LaBRI

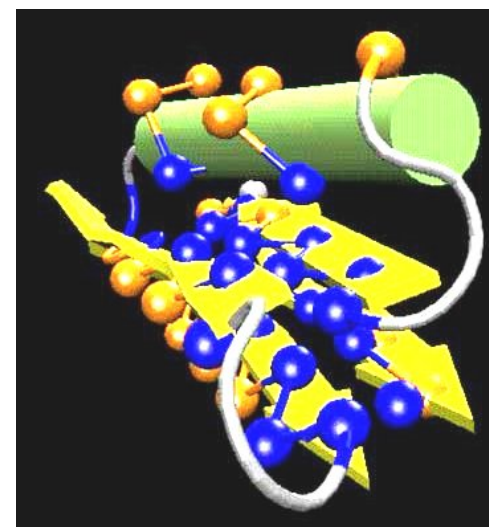
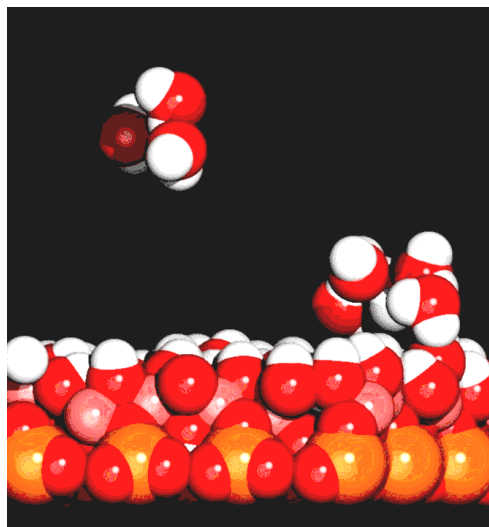
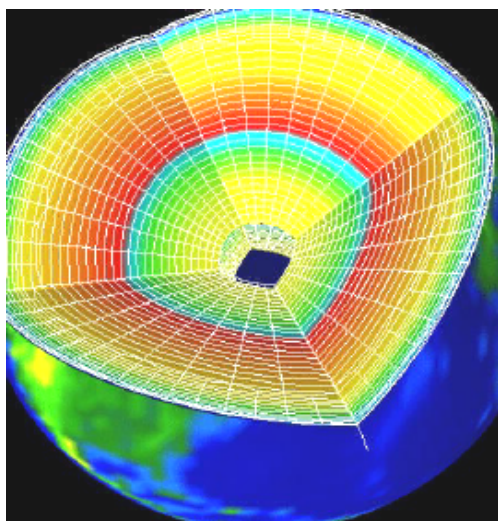
université
de **BORDEAUX**

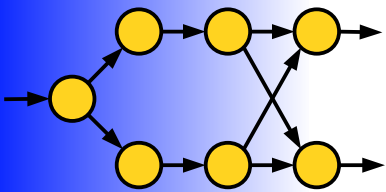
Inria



High-Performance Computing

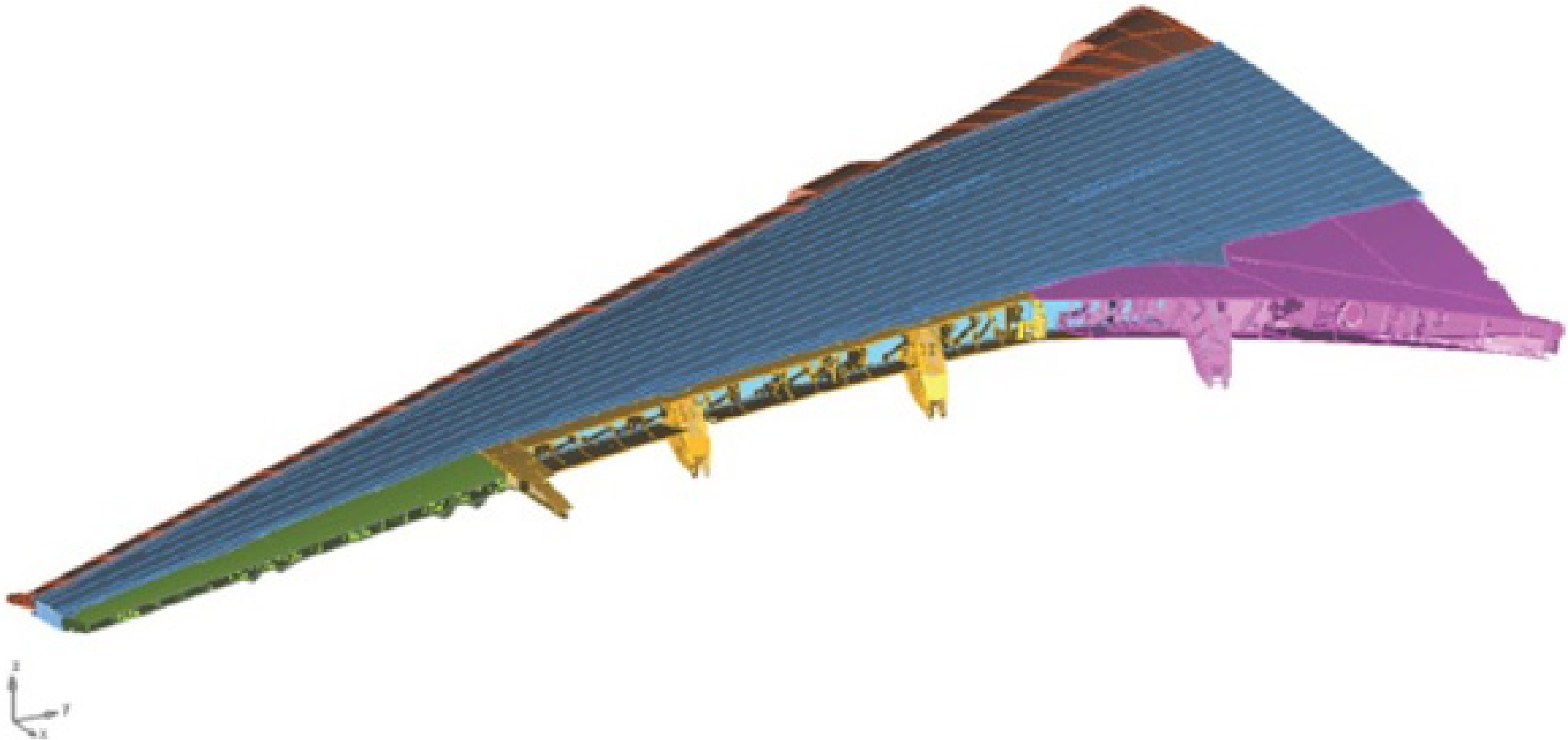
Simulation complements theory and experimentation



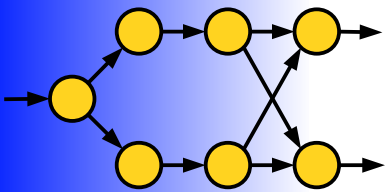


High-Performance Computing

Simulation complements theory and experimentation, e.g. A350

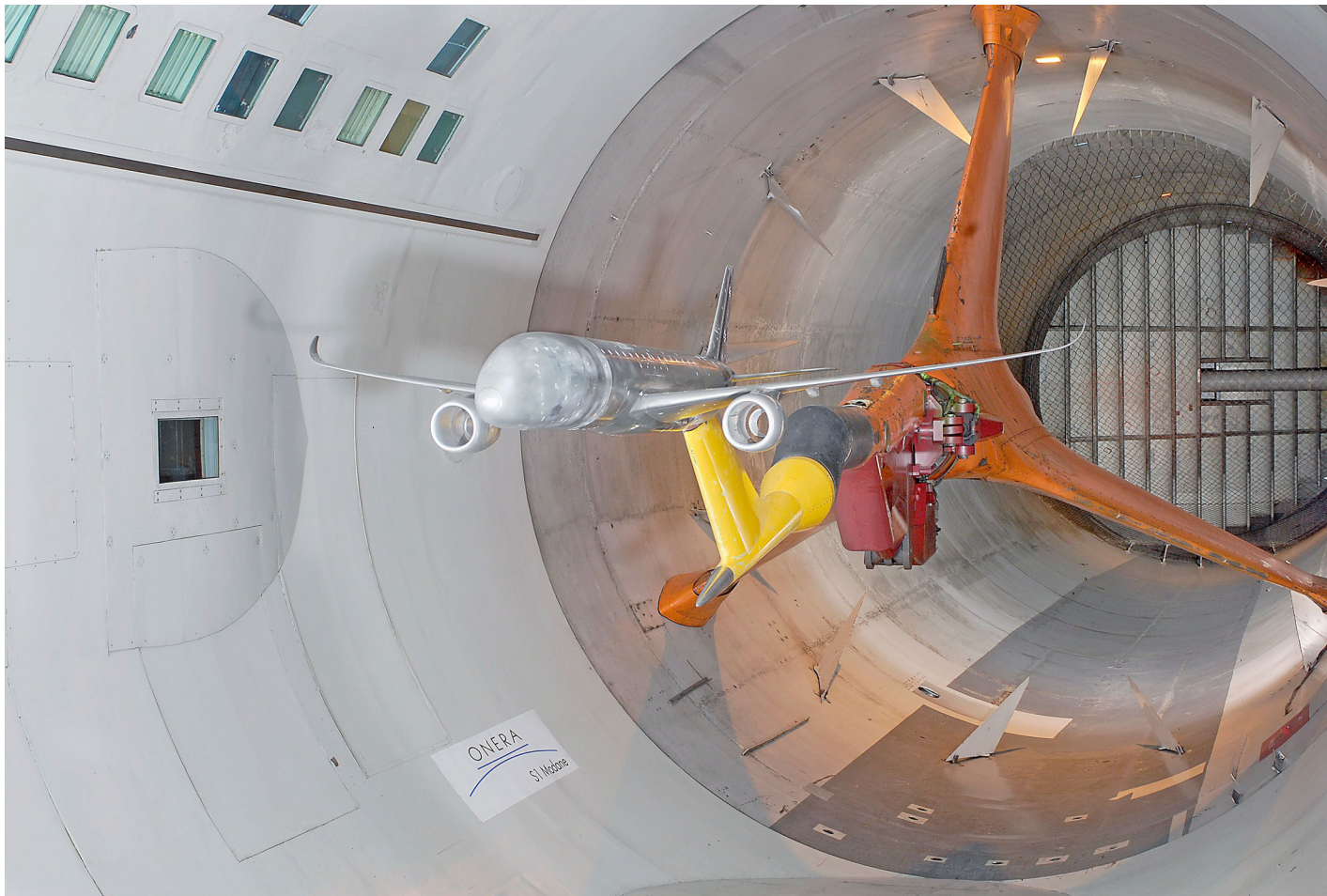


Simulated wing From Airbus

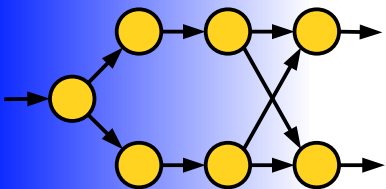


High-Performance Computing

Simulation complements theory and experimentation, e.g. A350



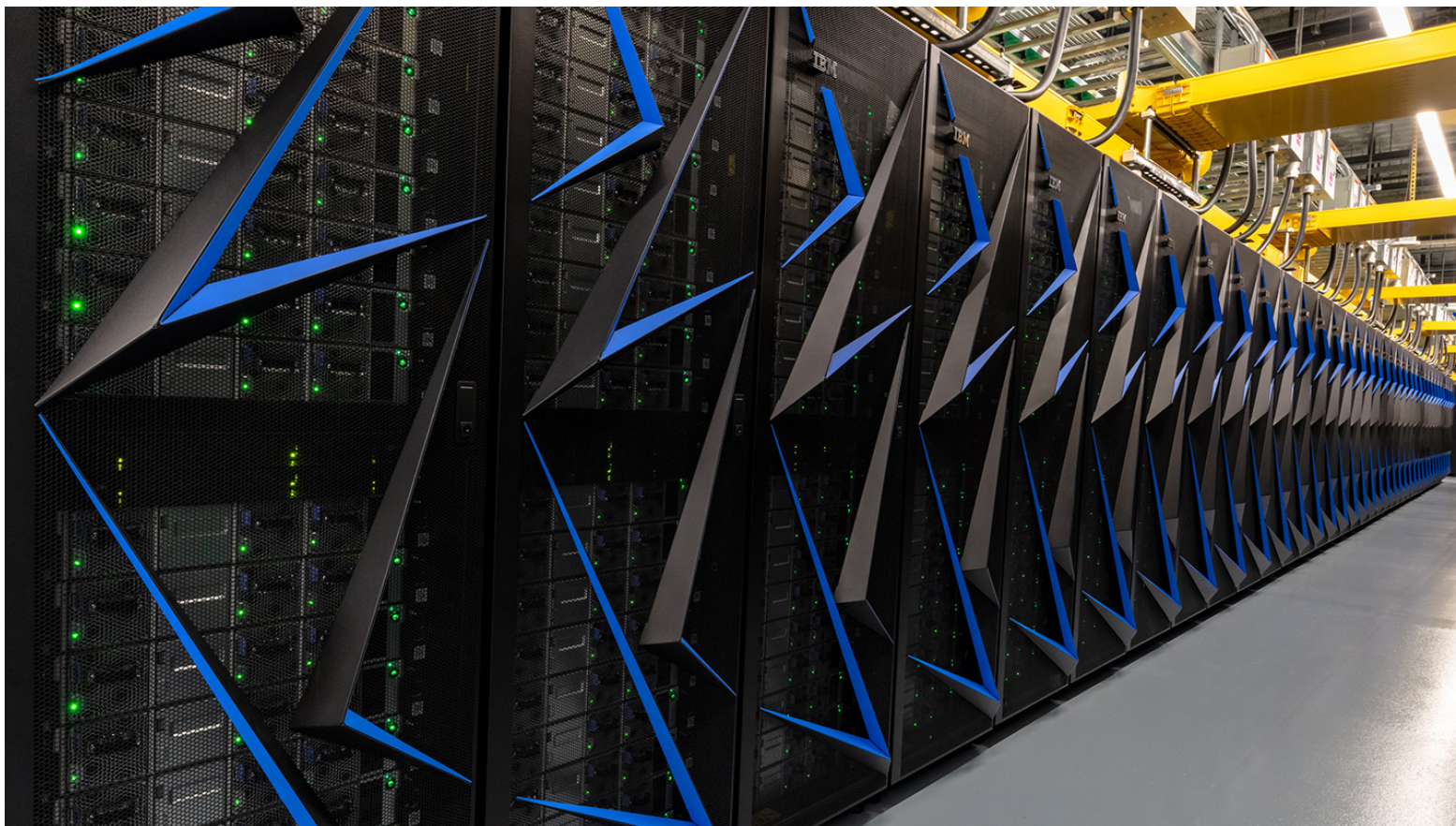
Wind tunnel from ONERA



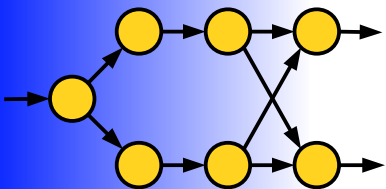
High-Performance Computing

Simulation complements theory and experimentation

But requires huge computational power



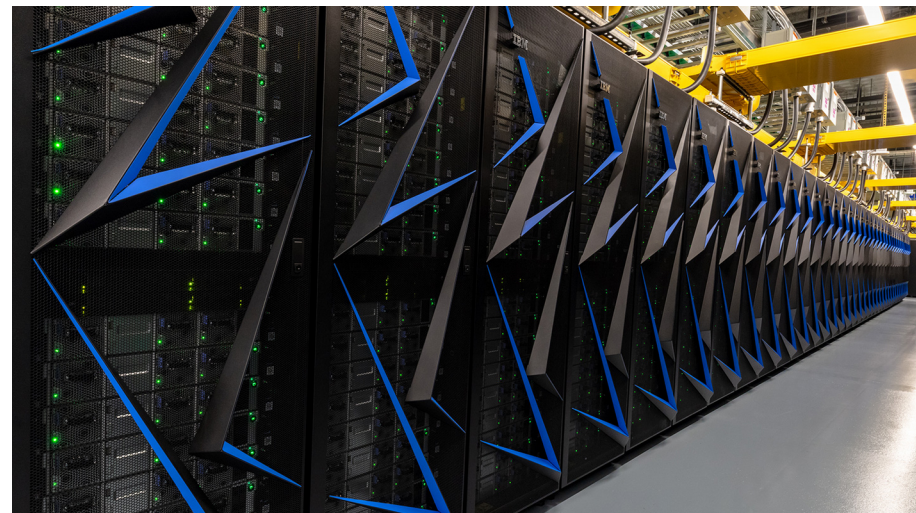
From ORNL



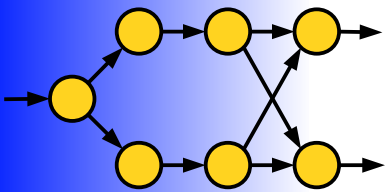
High-Performance Computing

Summit supercomputer

- #1 on LINPACK benchmark
- 143PF
- 4608 nodes
 - 2 IBM Power9 CPUs
 - 6 NVIDIA V100 GPUs
 - 512 GB DDR + 96 GB HBM2 mem
- IB 100G network
- 13MW



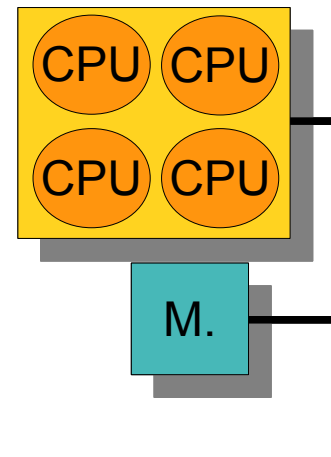
From ORNL

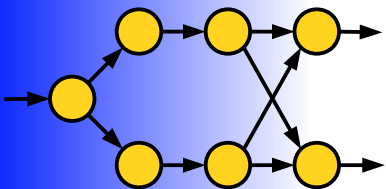


High-Performance Computing

Classical parallel programming

- threads

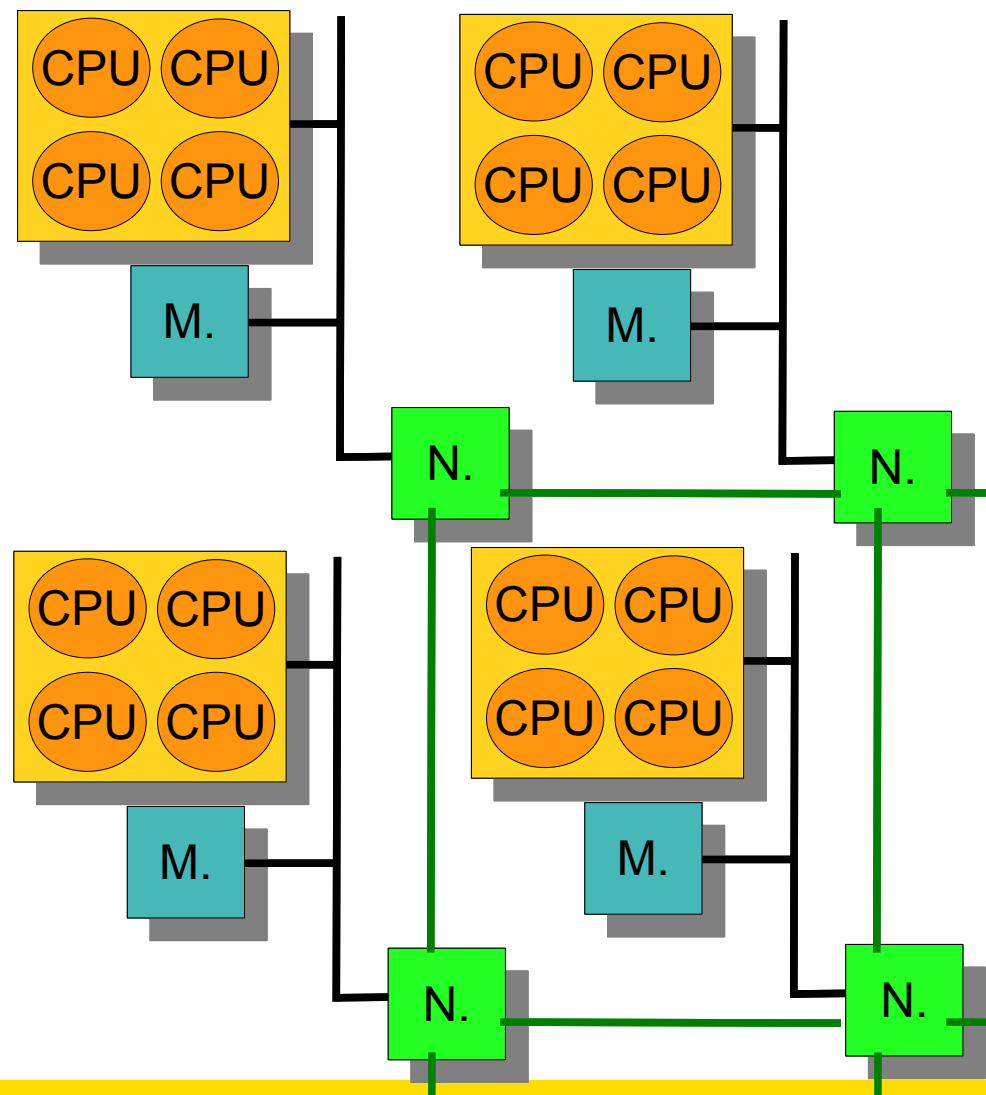


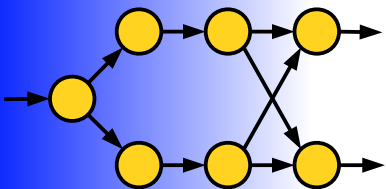


High-Performance Computing

Classical parallel programming

- threads
- MPI+threads



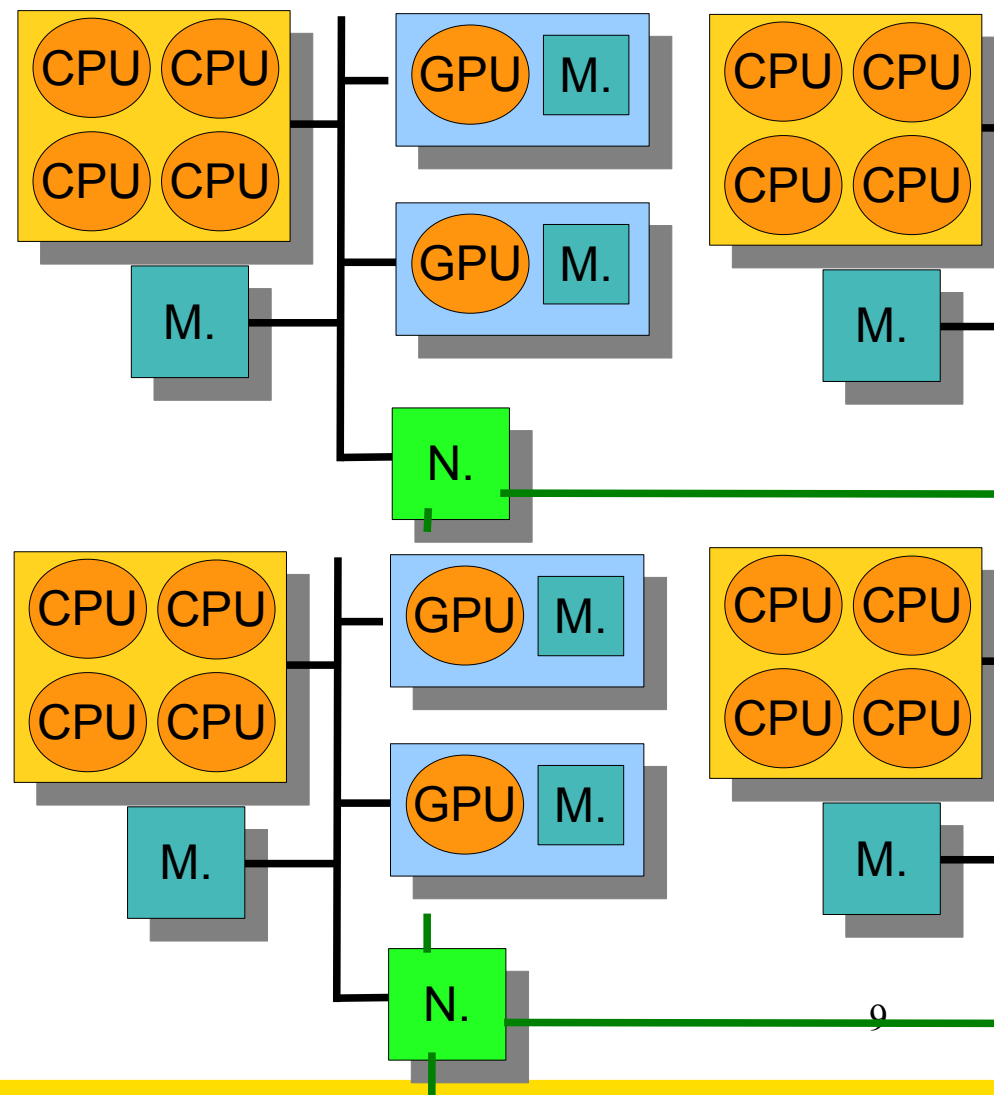


High-Performance Computing

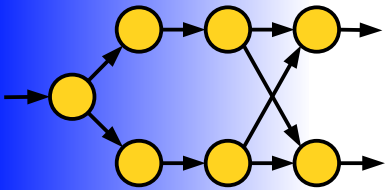
Classical parallel programming

- threads
- MPI+threads
- MPI+threads+CUDA

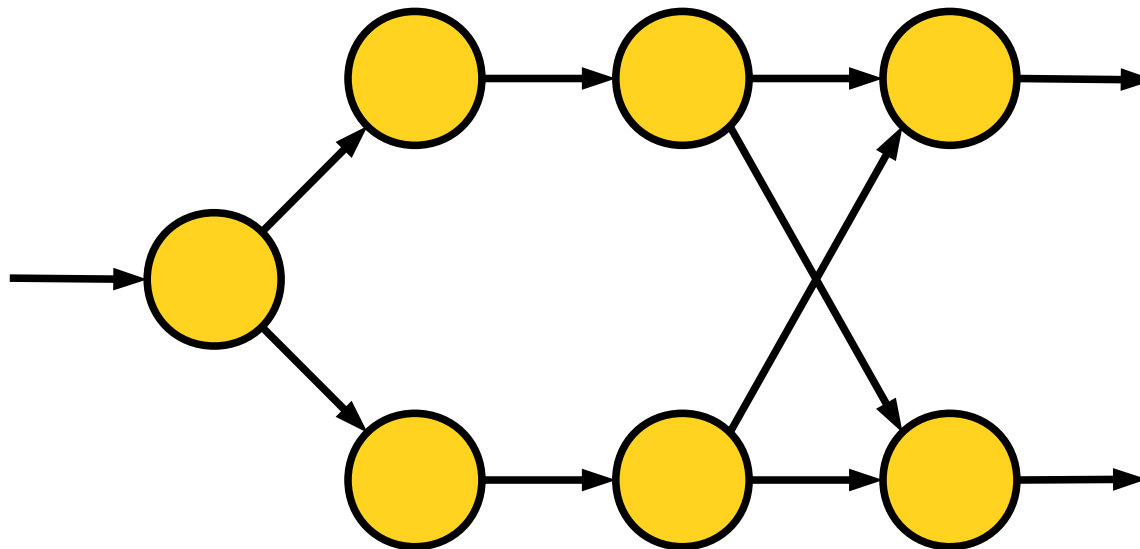
Managing interaction
between the three?



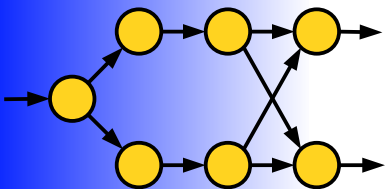
Task graphs



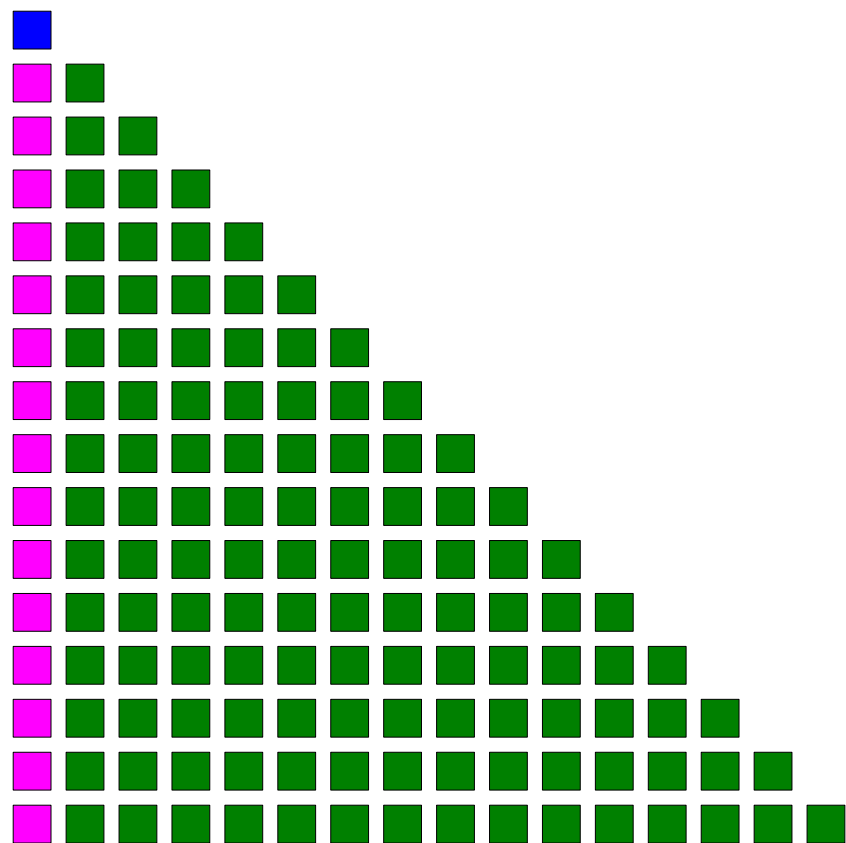
- Well-studied for scheduling parallelism (since '60s!)

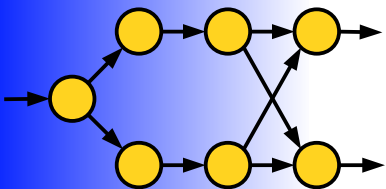


- Runtimes existing for a long time
 - e.g. Cilk, Athapascan-1 (end '90s)
- Until recently, not really actually used for scientific computation
 - Heterogeneous architectures pushed for them



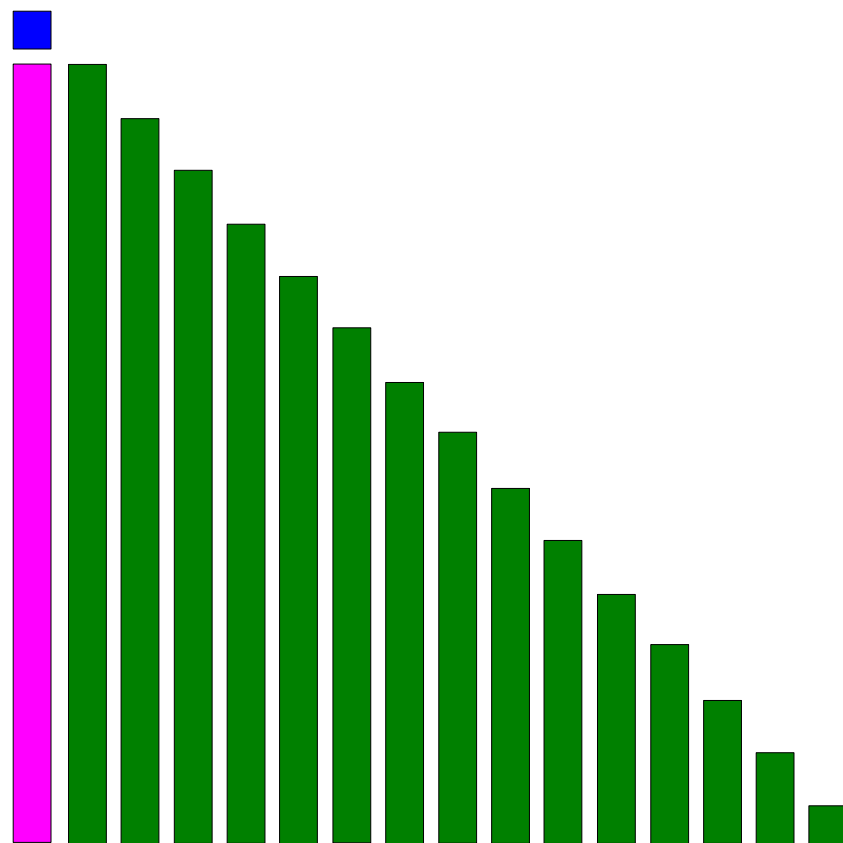
*PACK story

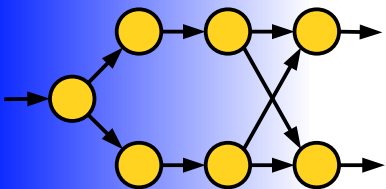




*PACK story

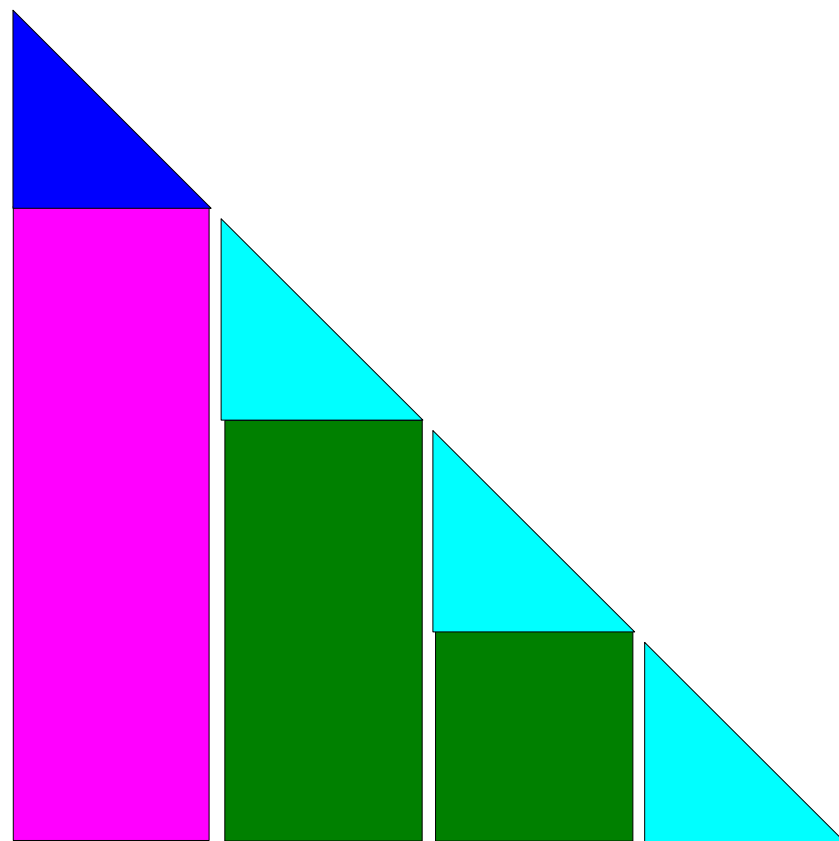
- LINPACK: vector computers (end '70s, '80s)

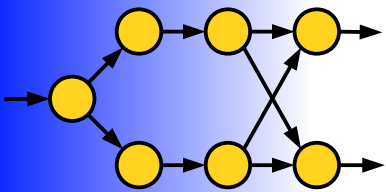




*PACK story

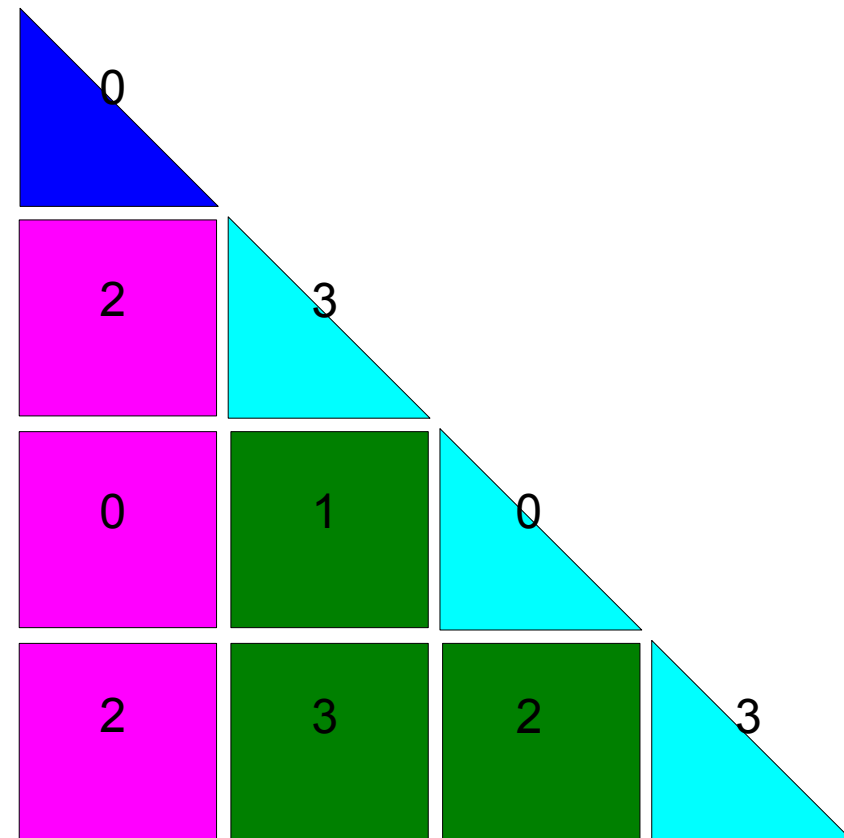
- LINPACK: vector computers (end '70s, '80s)
- LAPACK: cache-aware ('90s), blocked operation

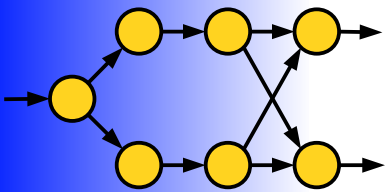




*PACK story

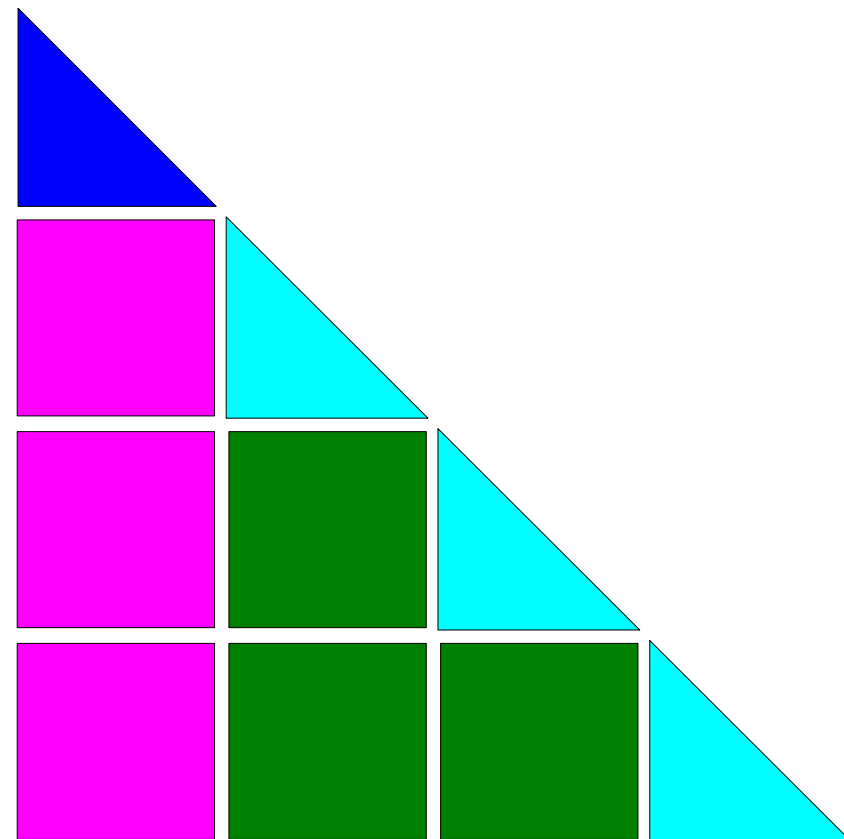
- LINPACK: vector computers (end '70s, '80s)
- LAPACK: cache-aware ('90s), blocked operation
- ScaLAPACK: distributed ('90s)
 - 2D block-cyclic distribution

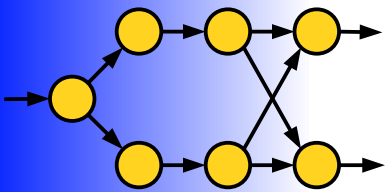




*PACK story

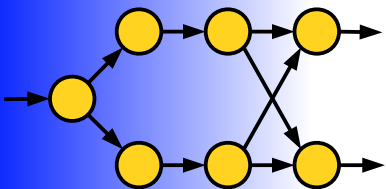
- LINPACK: vector computers (end '70s, '80s)
- LAPACK: cache-aware ('90s), blocked operation
- ScaLAPACK: distributed ('90s)
 - 2D block-cyclic distribution
- PLASMA: task graph (~'08)
 - + StarPU runtime
 - + MAGMA GPU kernels





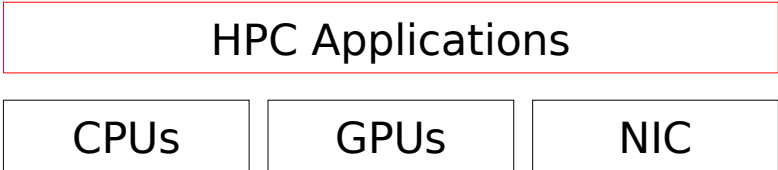
What happened?

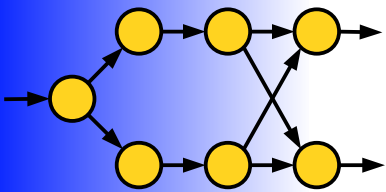
- Application writers wanted to keep control
 - Know their hardware
 - Strict ordering
 - Bulk Synchronous Parallelism
 - Hand-tuned pipelining
- Is that maintainable?
 - New hardware, rewrite?
- GPUs pushed to task graphs
 - MPI+threads+CUDA...



Runtime emerged naturally

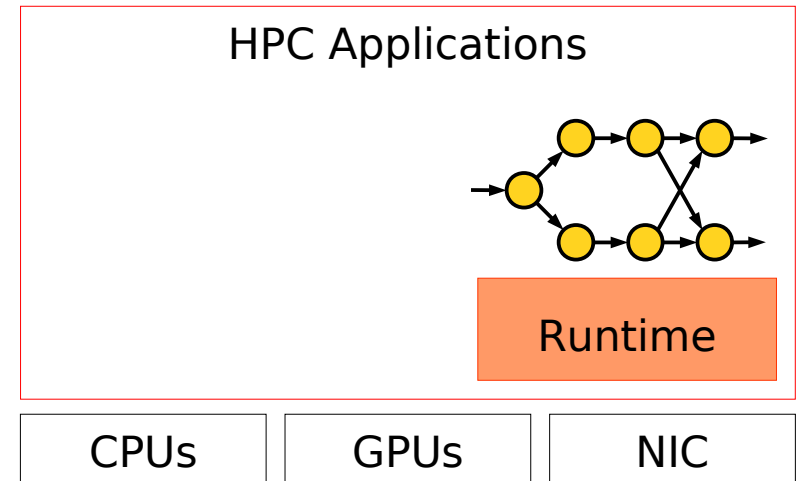
- From static control...

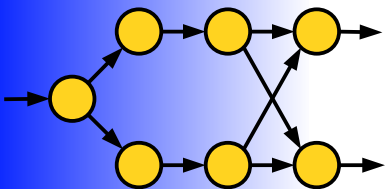




Runtime emerged naturally

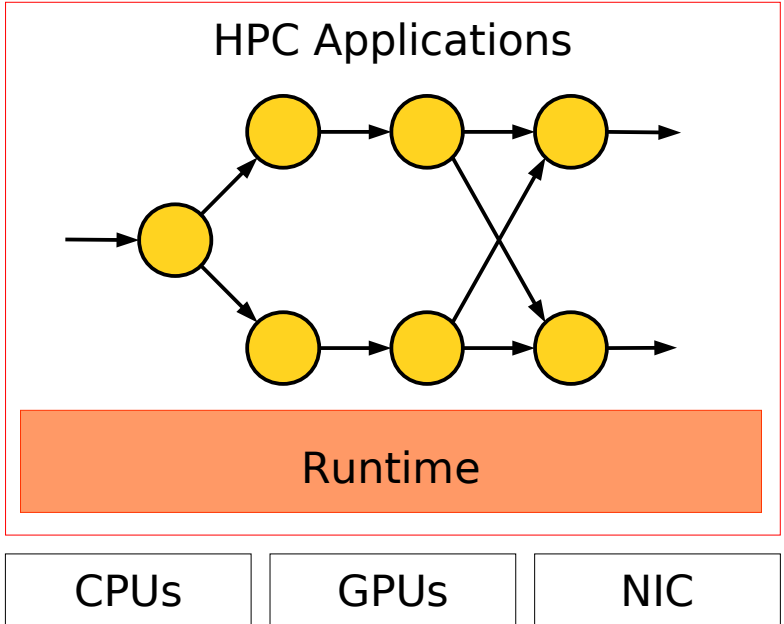
- From static control...
- To some dynamic control...

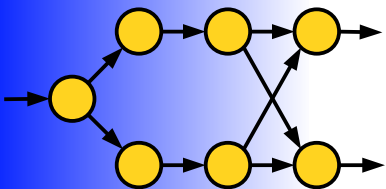




Runtime emerged naturally

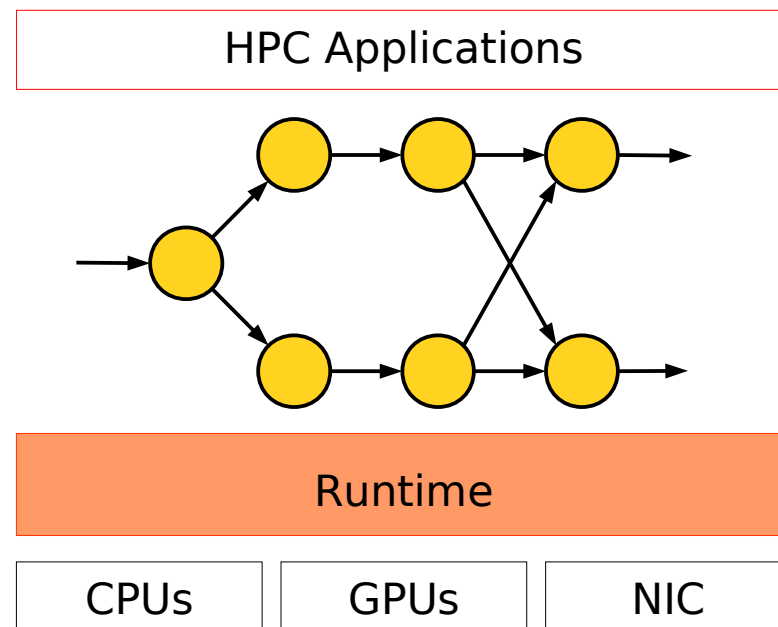
- From static control...
- To some dynamic control...
- To giving it all to the runtime...

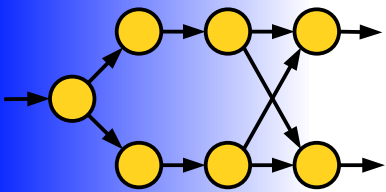




Runtime emerged naturally

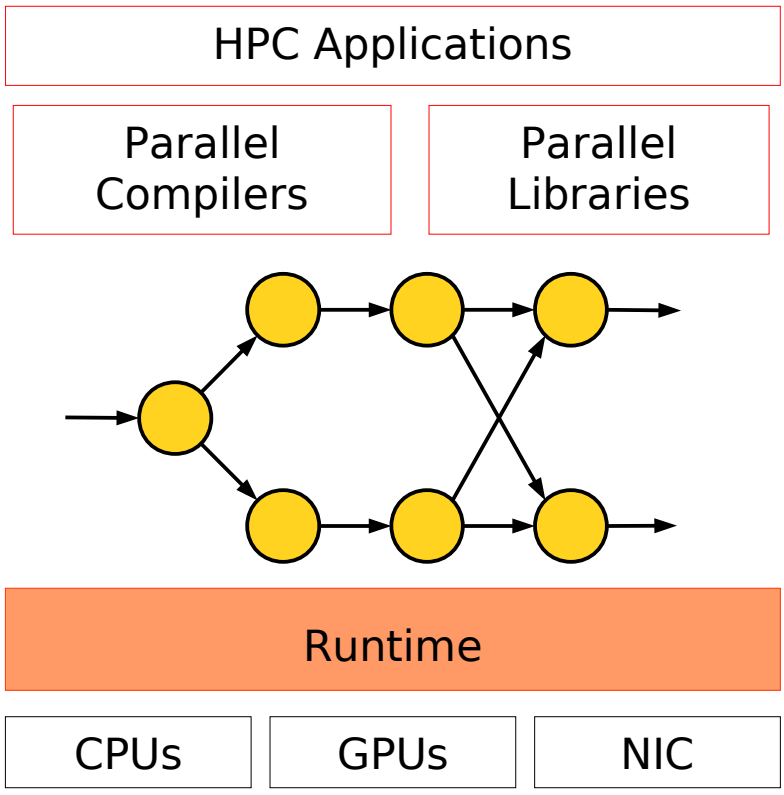
- From static control...
- To some dynamic control...
- To giving it all to the runtime...
- Or even an external runtime

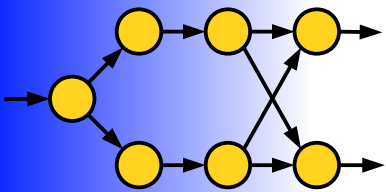




Runtime emerged naturally

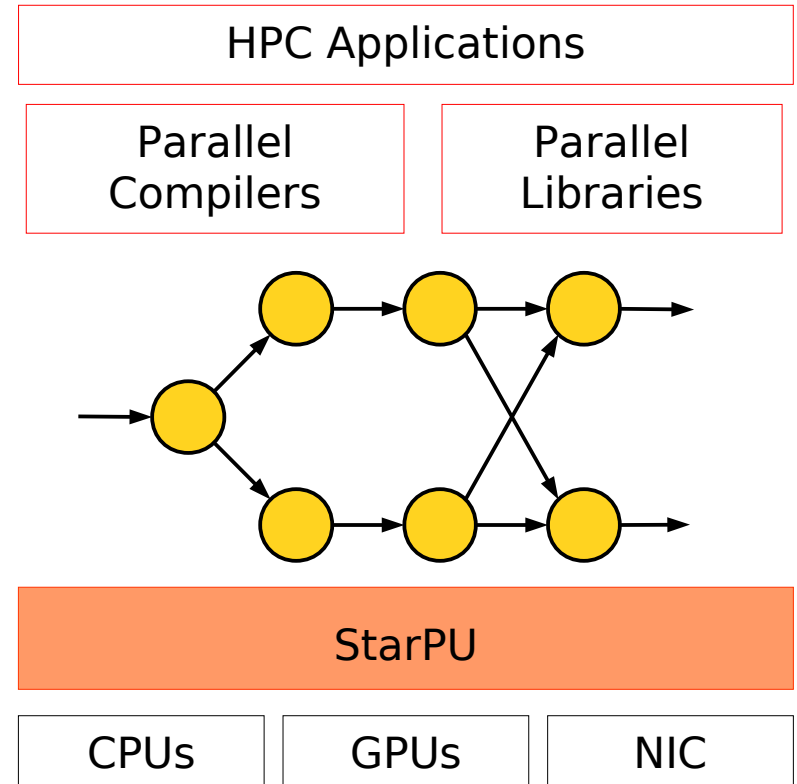
- From static control...
- To some dynamic control...
- To giving it all to the runtime...
- Or even an external runtime
- Possibly through intermediate layers
- Many projects leaned to this

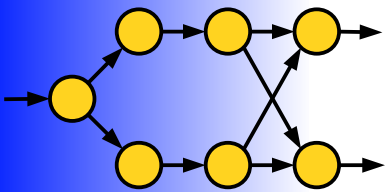




Runtime emerged naturally

- From static control...
- To some dynamic control...
- To giving it all to the runtime...
- Or even an external runtime
- Possibly through intermediate layers
- Many projects leaned to this
- StarPU from the start (2008) external
 - [AugonnetPhD11]





Expressing task graphs?

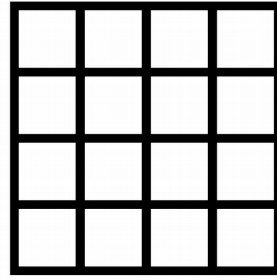
Sequential Task Flow (STF)





- Cholesky example from PLASMA

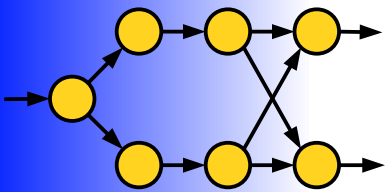
```

for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
           R,A[i][j], R,A[k][j]);
  }
}
task_wait_for_all();

```



	POTRF
	TRSM
	SYRK
	GEMM



Expressing task graphs?

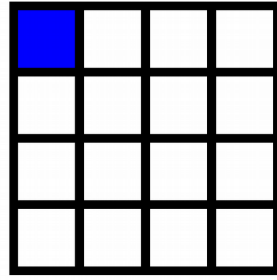
Sequential Task Flow (STF)





- Cholesky example from PLASMA

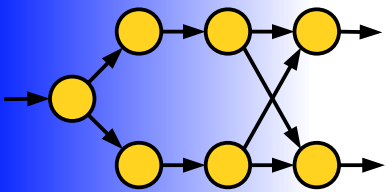
```

for (j = 0; j < N; j++) {
    POTRF (RW,A[j][j]);
    for (i = j+1; i < N; i++)
        TRSM (RW,A[i][j], R,A[j][j]);
    for (i = j+1; i < N; i++) {
        SYRK (RW,A[i][i], R,A[i][j]);
        for (k = j+1; k < i; k++)
            GEMM (RW,A[i][k],
                R,A[i][j], R,A[k][j]);
    }
}
task_wait_for_all();

```



	POTRF
	TRSM
	SYRK
	GEMM



Expressing task graphs?

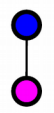
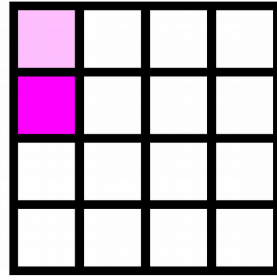
Sequential Task Flow (STF)





- Cholesky example from PLASMA

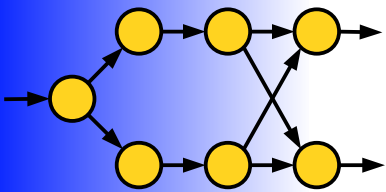
```

for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
           R,A[i][j], R,A[k][j]);
  }
}
task_wait_for_all();

```



	POTRF
	TRSM
	SYRK
	GEMM



Expressing task graphs?

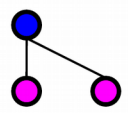
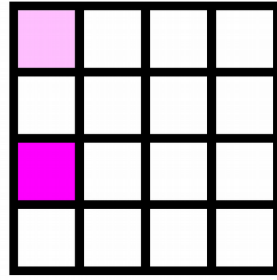
Sequential Task Flow (STF)





- Cholesky example from PLASMA

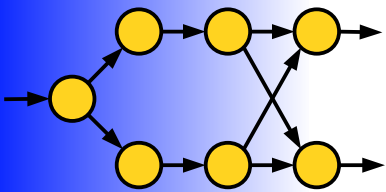
```

for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
task_wait_for_all();

```



	POTRF
	TRSM
	SYRK
	GEMM



Expressing task graphs?

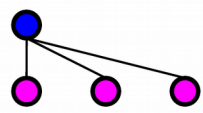
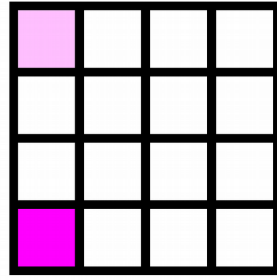
Sequential Task Flow (STF)





- Cholesky example from PLASMA

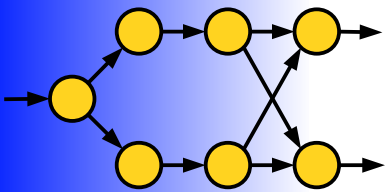
```

for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
           R,A[i][j], R,A[k][j]);
  }
}
task_wait_for_all();

```



	POTRF
	TRSM
	SYRK
	GEMM



Expressing task graphs?

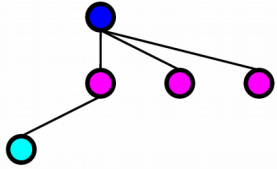
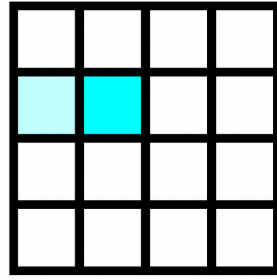
Sequential Task Flow (STF)





- Cholesky example from PLASMA

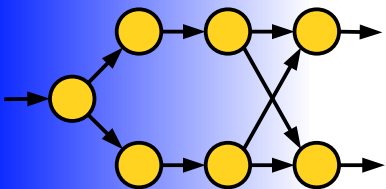
```

for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
task_wait_for_all();

```



	POTRF
	TRSM
	SYRK
	GEMM



Expressing task graphs?

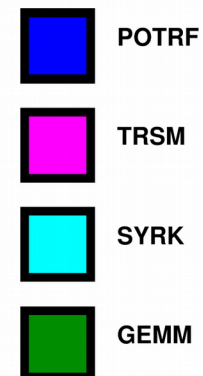
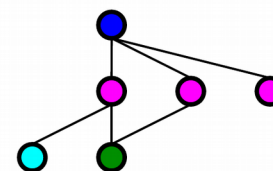
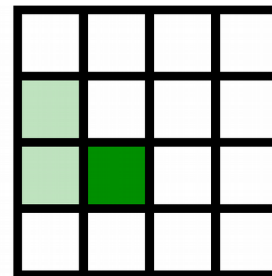
Sequential Task Flow (STF)

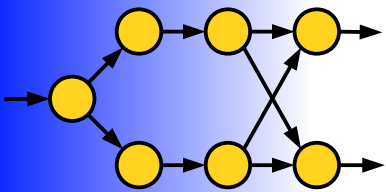
- Cholesky example from PLASMA

```

for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
           R,A[i][j], R,A[k][j]);
  }
}
task_wait_for_all();

```





Expressing task graphs?

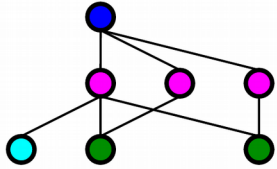
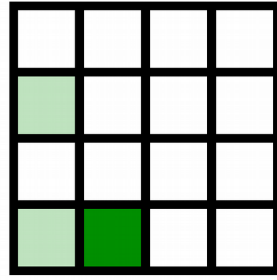
Sequential Task Flow (STF)





- Cholesky example from PLASMA

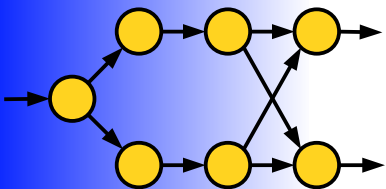
```

for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
           R,A[i][j], R,A[k][j]);
  }
}
task_wait_for_all();

```



	POTRF
	TRSM
	SYRK
	GEMM



Expressing task graphs?

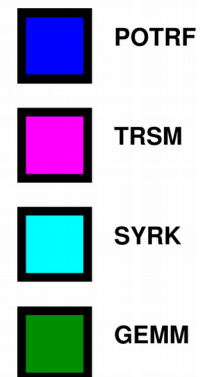
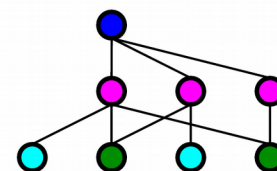
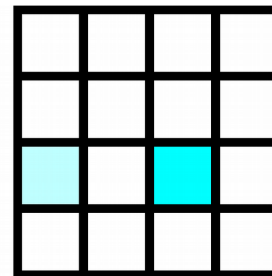
Sequential Task Flow (STF)

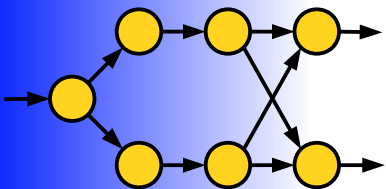
- Cholesky example from PLASMA

```

for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
           R,A[i][j], R,A[k][j]);
  }
}
task_wait_for_all();

```





Expressing task graphs?

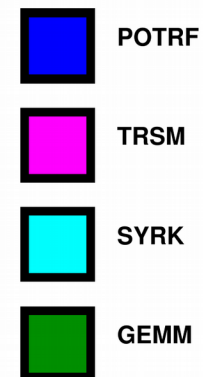
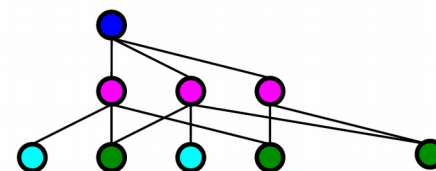
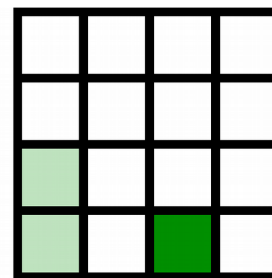
Sequential Task Flow (STF)

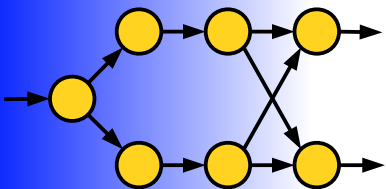
- Cholesky example from PLASMA

```

for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
           R,A[i][j], R,A[k][j]);
  }
}
task_wait_for_all();

```





Expressing task graphs?

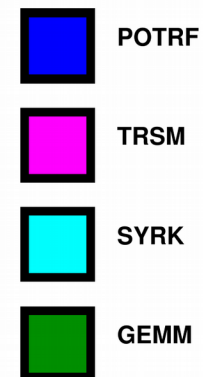
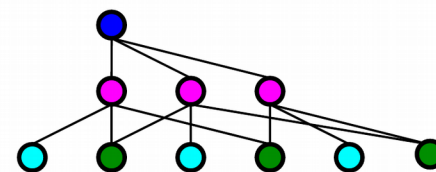
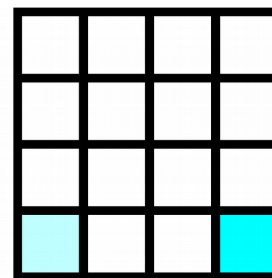
Sequential Task Flow (STF)

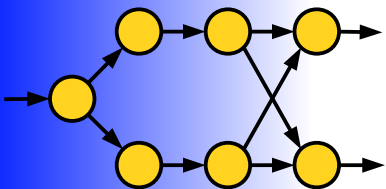
- Cholesky example from PLASMA

```

for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
task_wait_for_all();

```





Expressing task graphs?

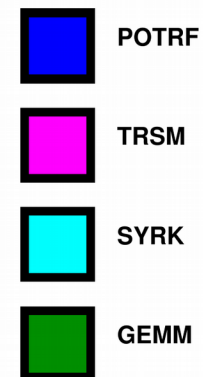
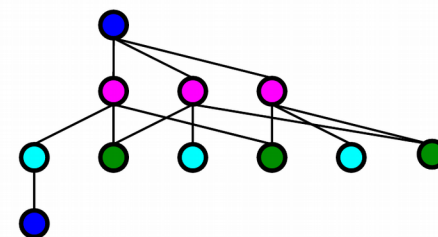
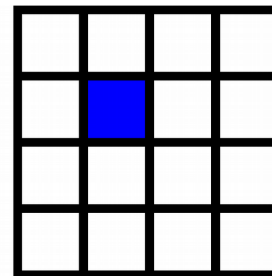
Sequential Task Flow (STF)

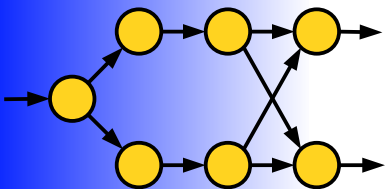
- Cholesky example from PLASMA

```

for (j = 0; j < N; j++) {
  POTRF (RW, A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW, A[i][j], R, A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW, A[i][i], R, A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW, A[i][k],
           R, A[i][j], R, A[k][j]);
  }
}
task_wait_for_all();

```





Expressing task graphs?

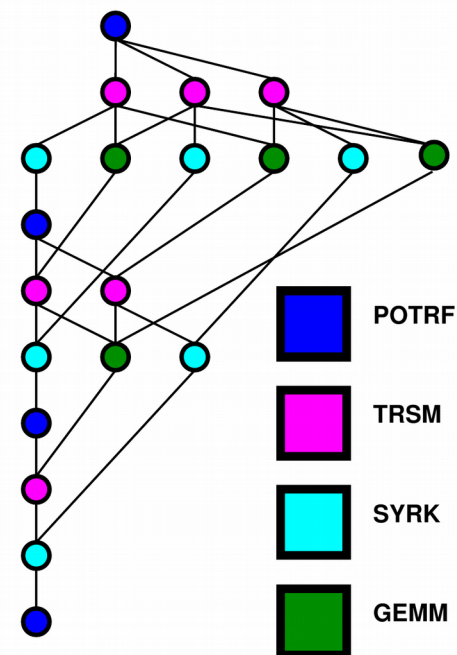
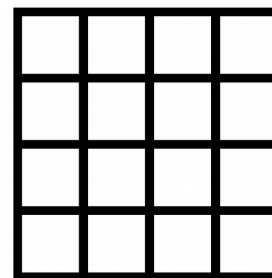
Sequential Task Flow (STF)

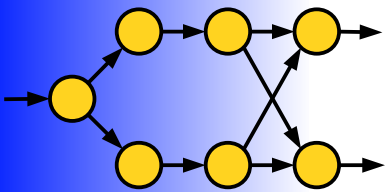
- Cholesky example from PLASMA

```

for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
           R,A[i][j], R,A[k][j]);
  }
}
task_wait_for_all();

```



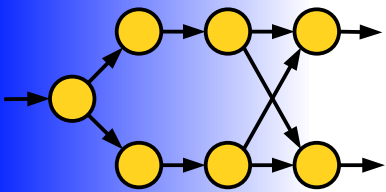


Expressing task graphs?

Sequential Task Flow (STF)

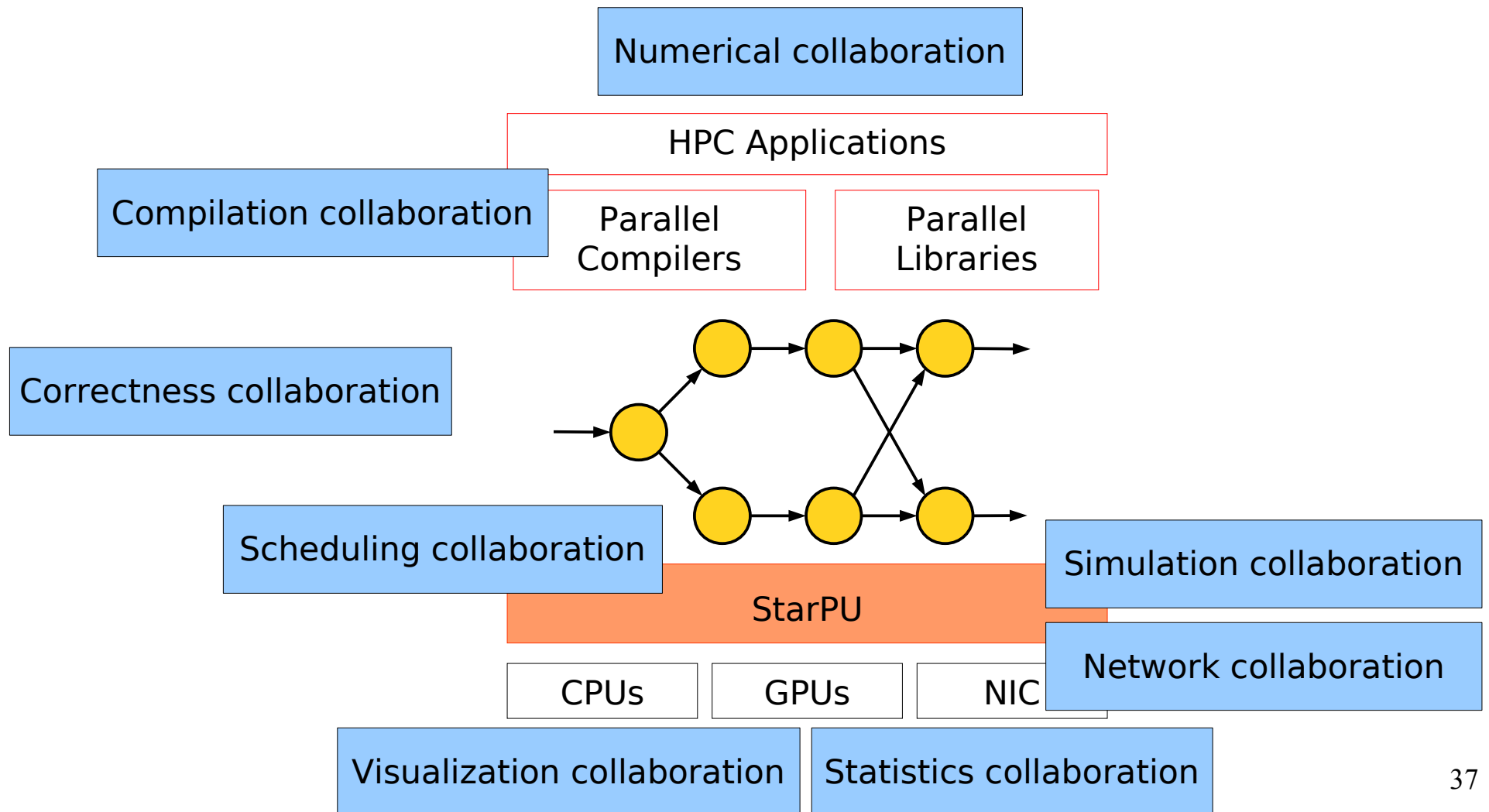
- Sequential-looking source code
 - Just expresses the algorithm
- Can debug sequential version
- Runtime will handle parallel execution

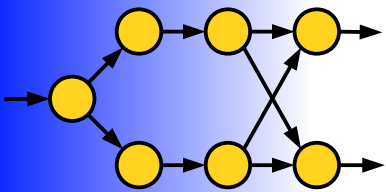
```
for (j = 0; j < N; j++) {
    POTRF (RW,A[j][j]);
    for (i = j+1; i < N; i++)
        TRSM (RW,A[i][j], R,A[j][j]);
    for (i = j+1; i < N; i++) {
        SYRK (RW,A[i][i], R,A[i][j]);
        for (k = j+1; k < i; k++)
            GEMM (RW,A[i][k],
                 R,A[i][j], R,A[k][j]);
    }
}
task_wait_for_all();
```



Task graphs as central notion

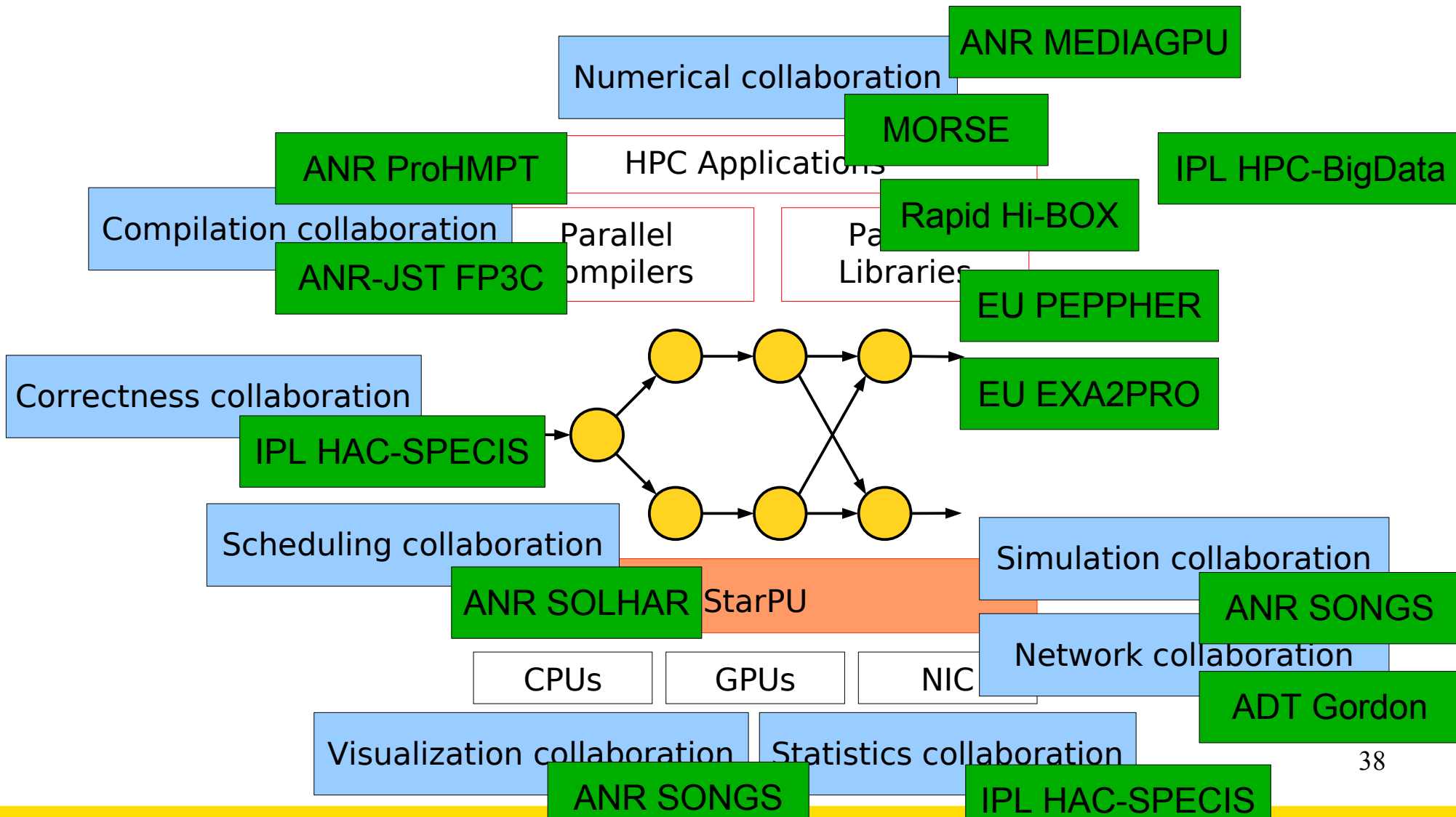
Task graphs as a bridge between various expertise

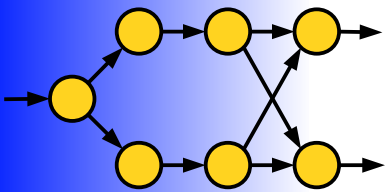




Task graphs as central notion

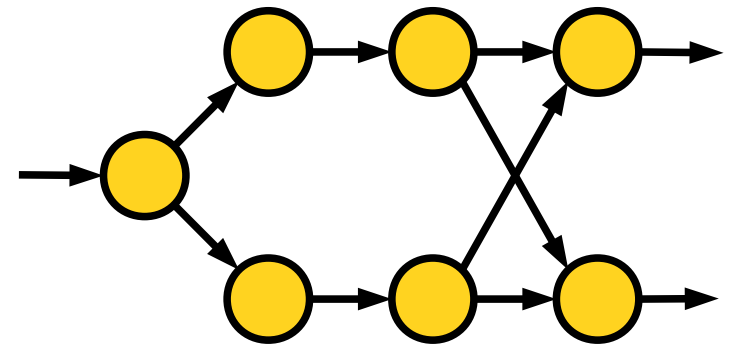
Task graphs as a bridge between various expertise

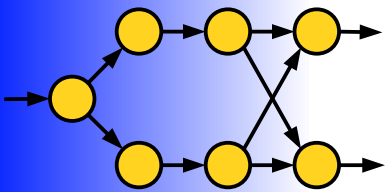




Today's agenda

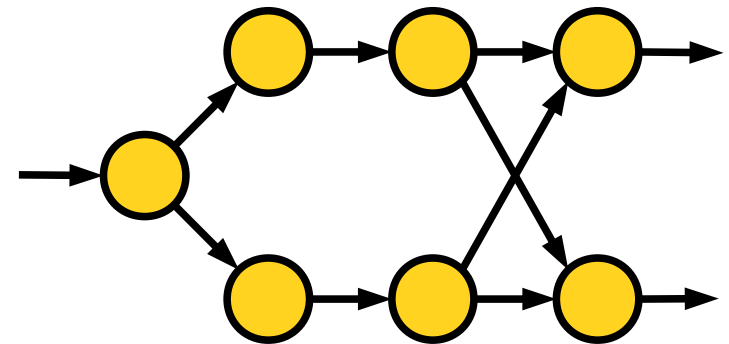
- Task graph scheduling
 - What is a runtime scheduler?
 - Beyond classical HEFT
 - Involving theoreticians
- Dividable tasks
- Distributed execution

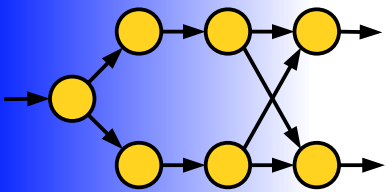




Today's agenda

- **Task graph scheduling**
 - What is a runtime scheduler?
 - Beyond classical HEFT
 - Involving theoreticians
- **Dividable tasks**
- **Distributed execution**

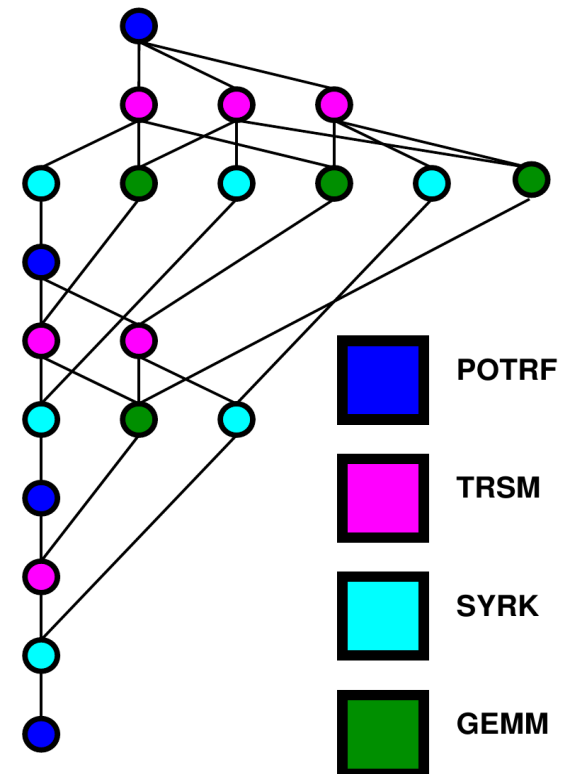


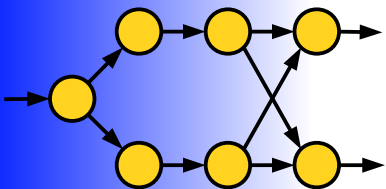


Task graph scheduling

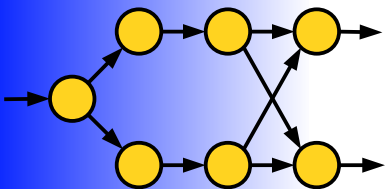
Target: completion time, usually (could also be energy)

- Care of critical path
 - Task priorities
- Care of leveraging accelerators
 - Task duration
- Care of data transfers
 - Transfer penalty



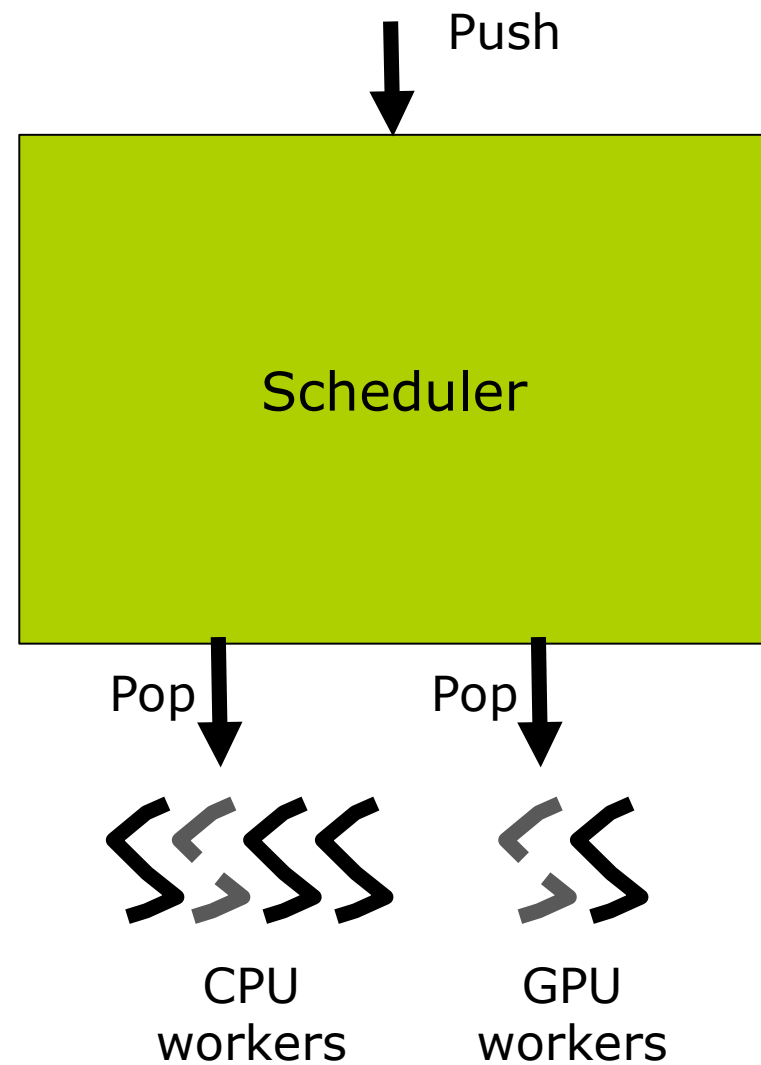


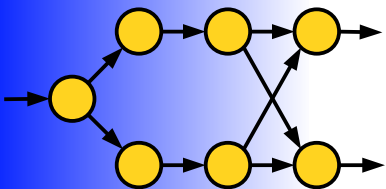
What is a runtime scheduler?



What is a runtime scheduler?

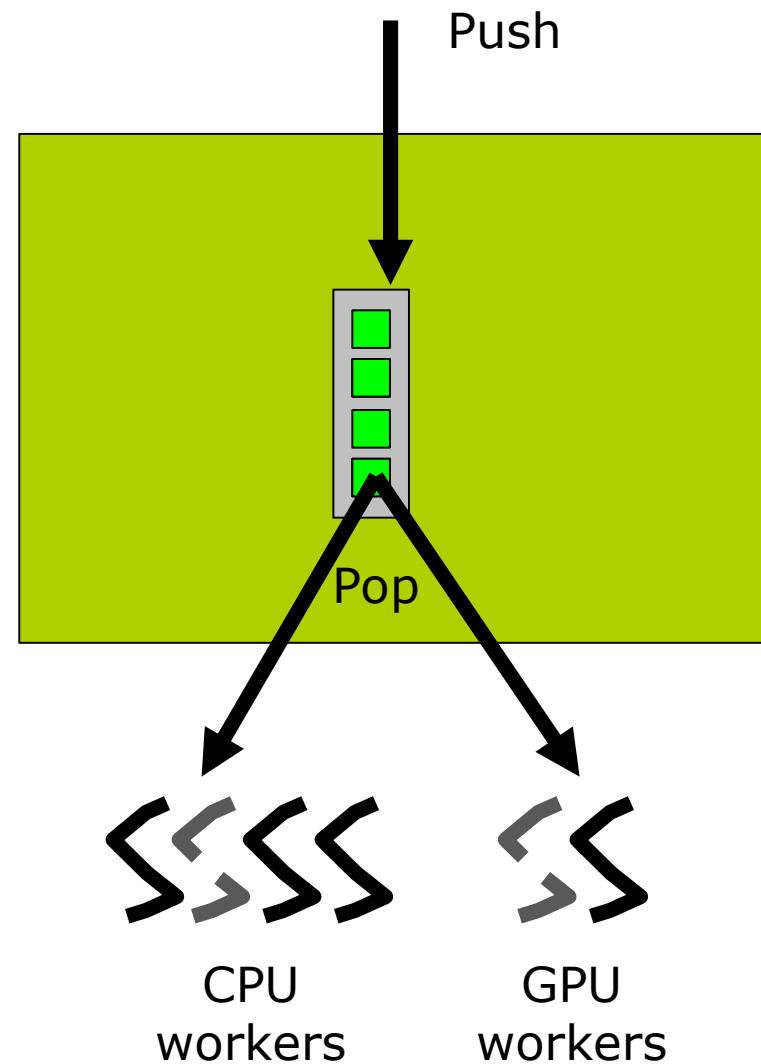
- Ready tasks

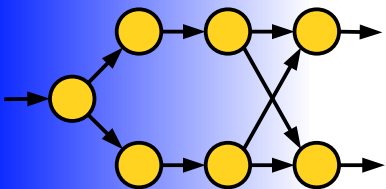




What is a runtime scheduler?

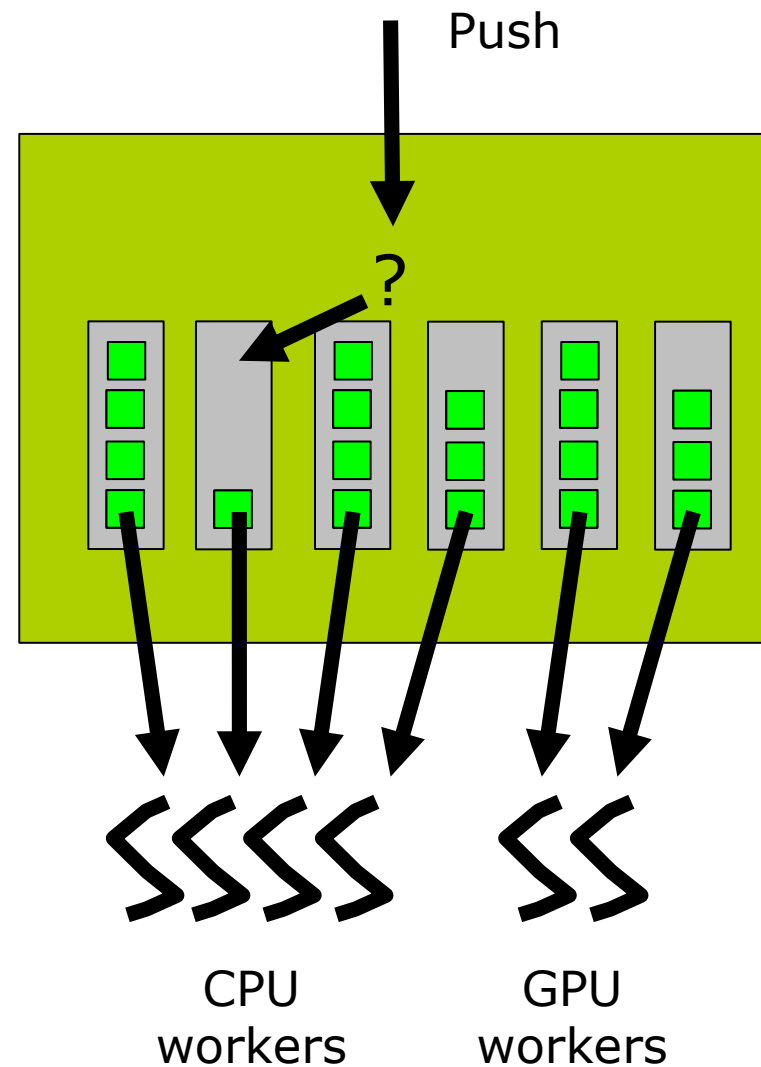
- Ready tasks
- Single eager list

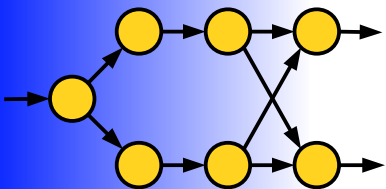




What is a runtime scheduler?

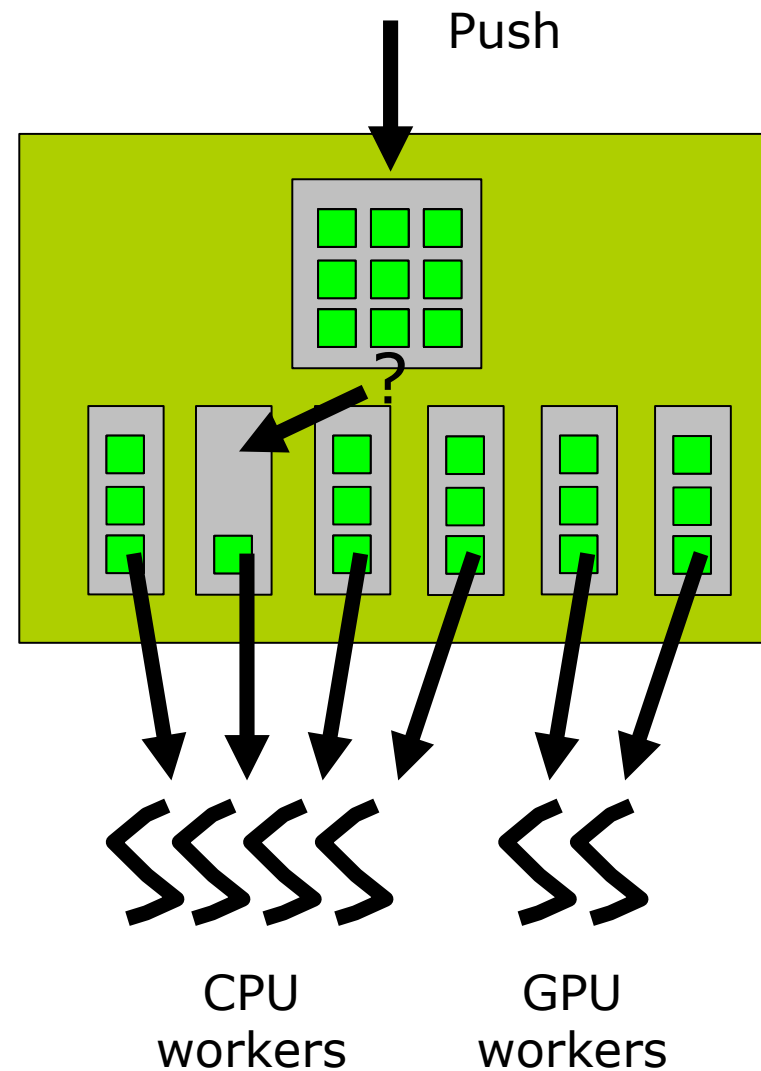
- Ready tasks
- Single eager list
- Early decision

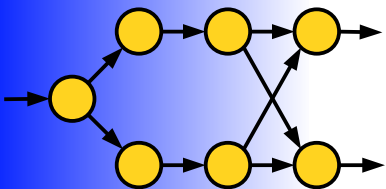




What is a runtime scheduler?

- Ready tasks
- Single eager list
- Early decision
- But not too early



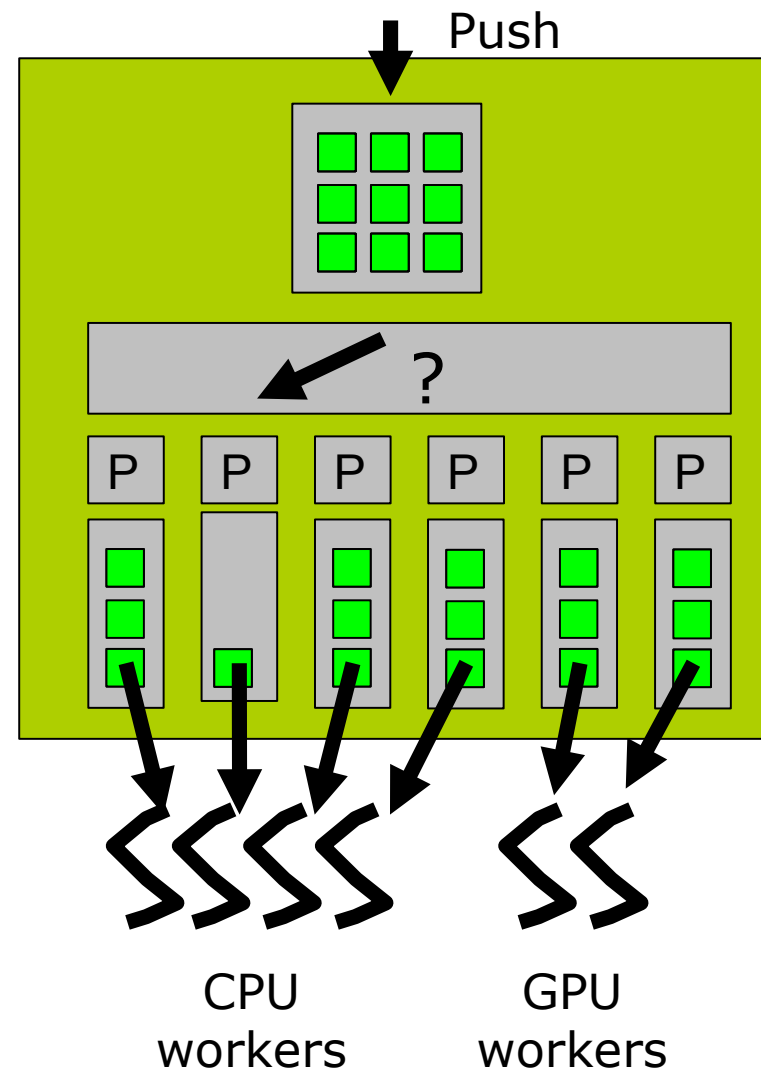


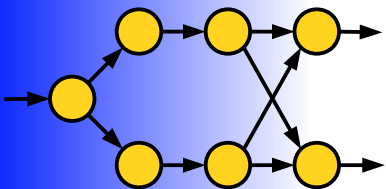
What is a runtime scheduler?

- Ready tasks
- Single eager list
- Early decision
- But not too early

Move to component-based

- [Archipoff13, SergentPhD16]
- Tasks pushed/pulled



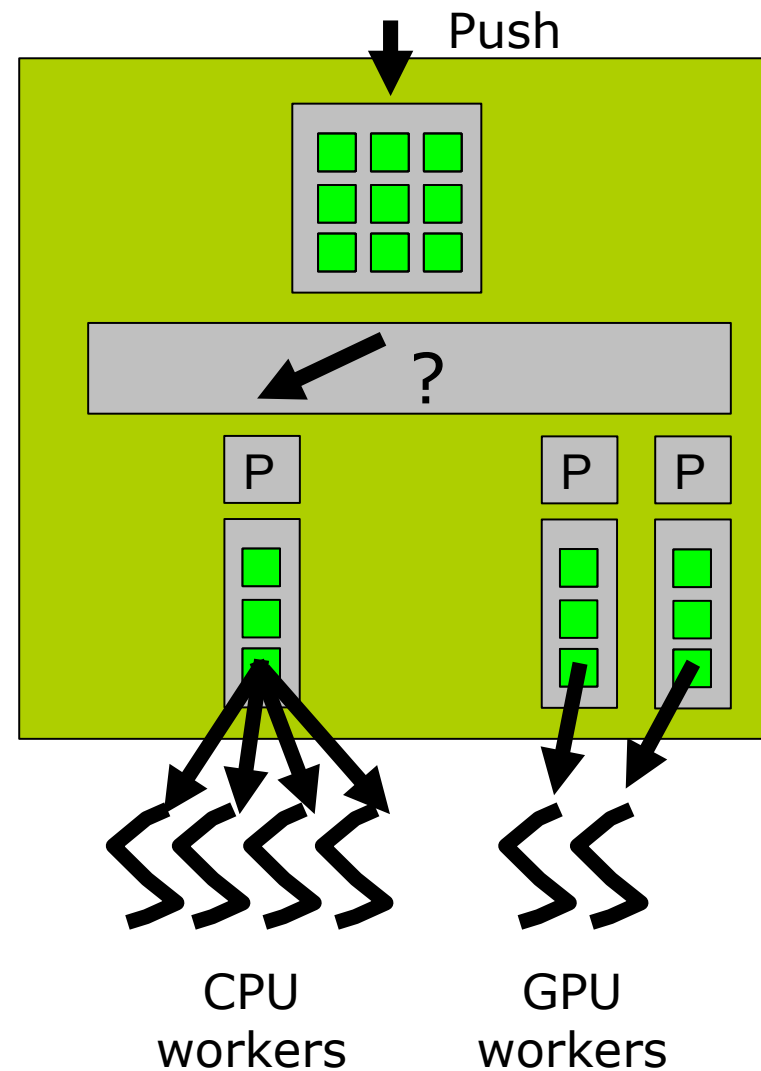


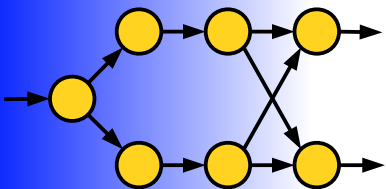
What is a runtime scheduler?

- Ready tasks
- Single eager list
- Early decision
- But not too early

Move to component-based

- [Archipoff13, SergentPhD16]
- Tasks pushed/pulled



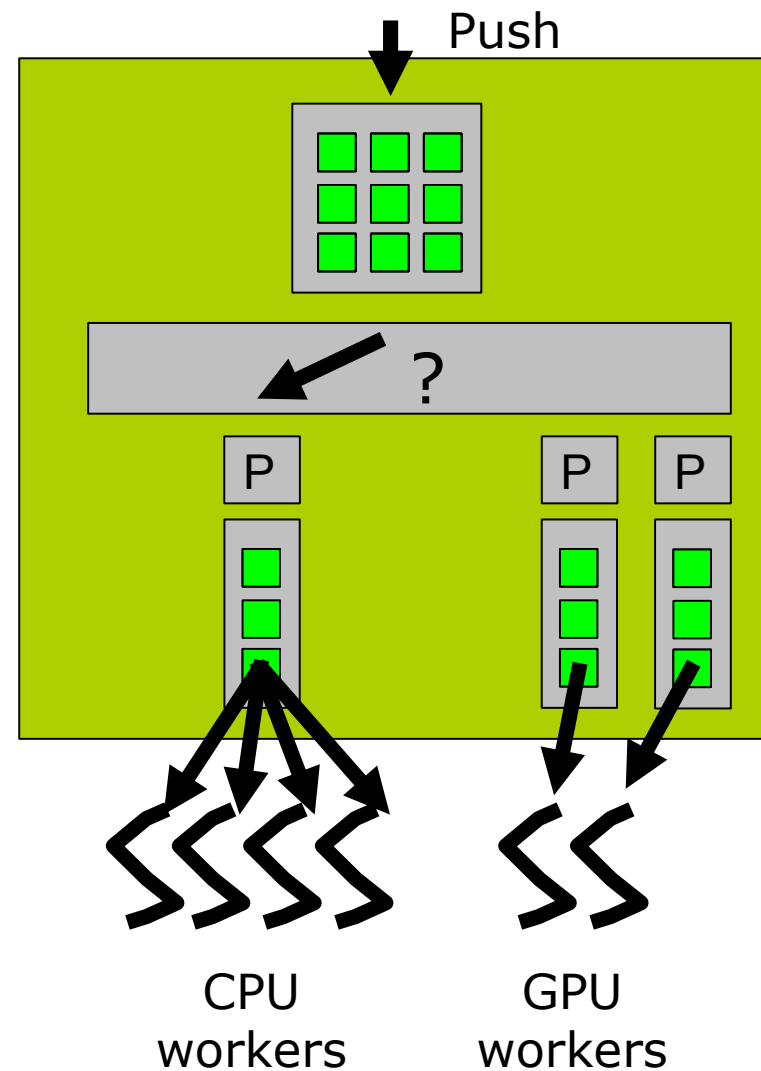


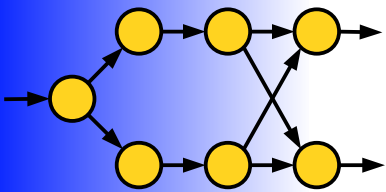
What is a runtime scheduler?

- Ready tasks
- Single eager list
- Early decision
- But not too early

Move to component-based

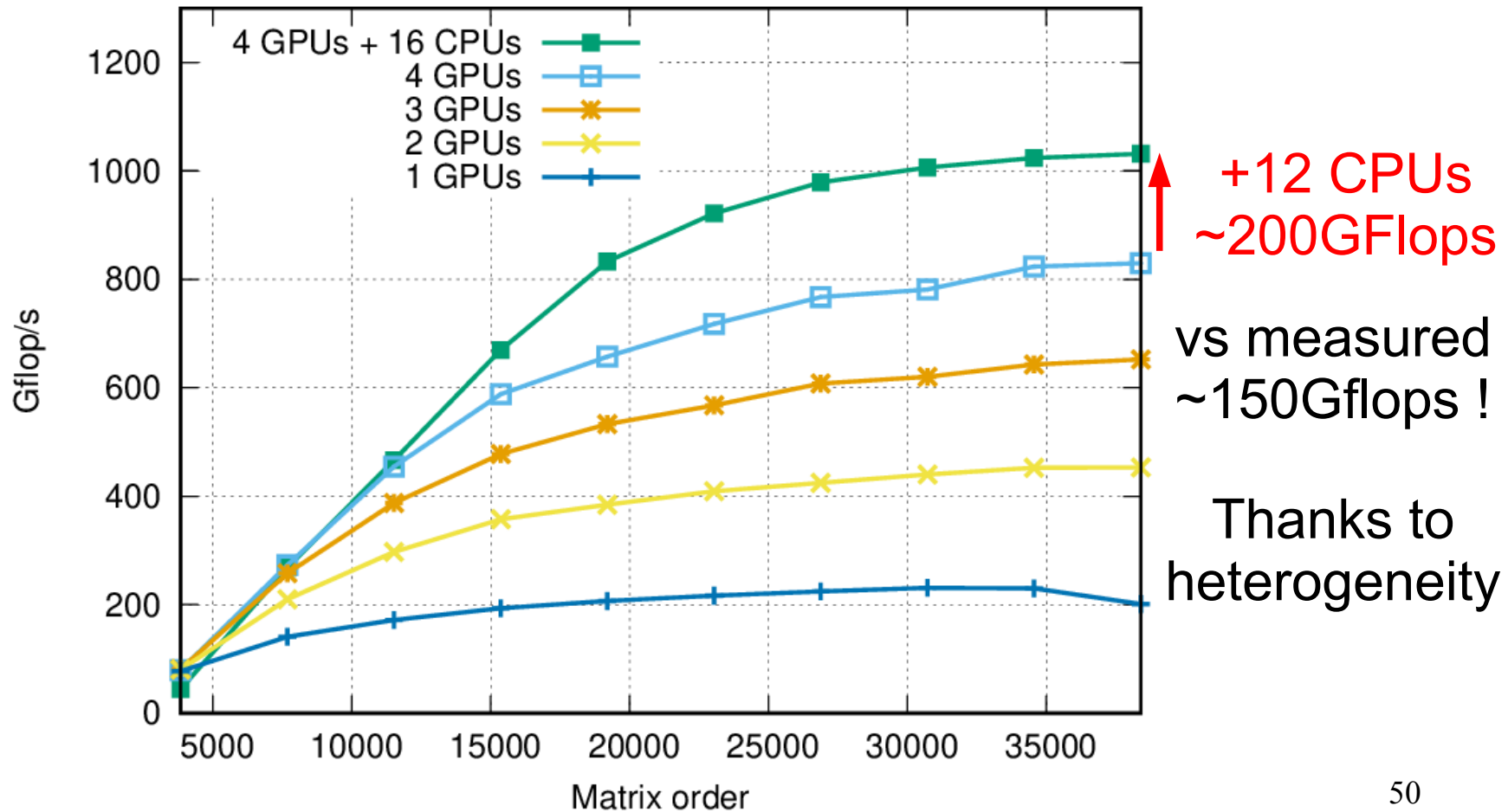
- [Archipoff13, SergentPhD16]
- Tasks pushed/pulled
- Would welcome correctness model

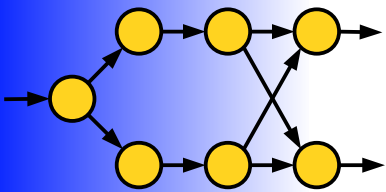




Typical obtained performance

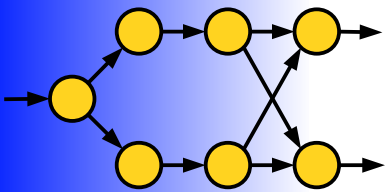
- With classical HEFT-like scheduling heuristic (dmdas)
 - e.g. QR factorization





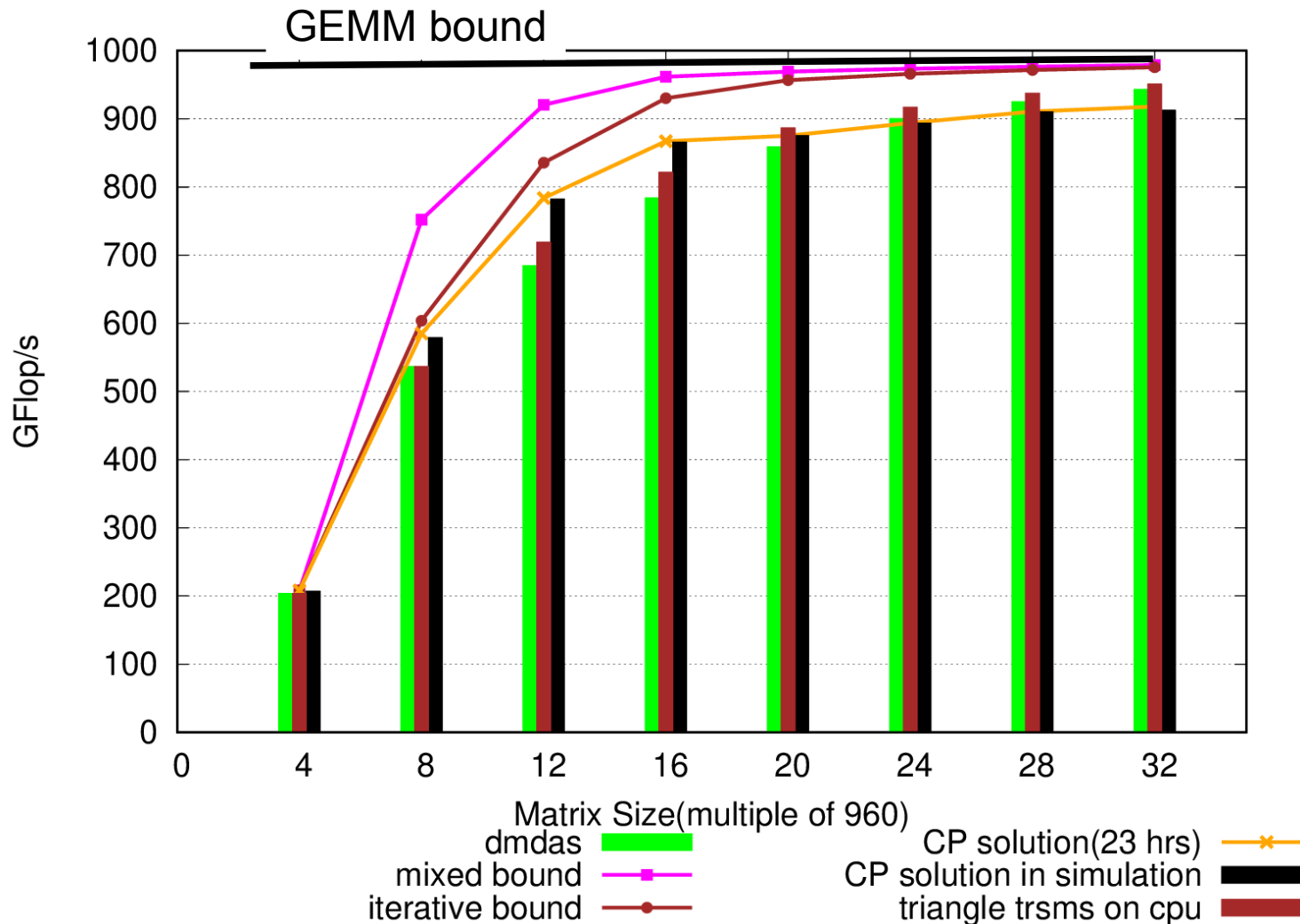
Scheduling beyond state of the art

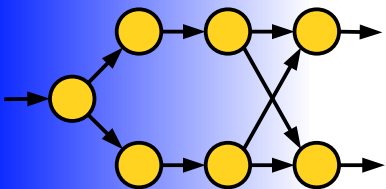
- HEFT-like heuristic dmdas is strong
 - But can do better
- [KumarPhD17], co-advised with
 - Theoreticians O. Beaumont, L. Eyraud-Dubois
 - Numerical analyst E. Agullo
 - Runtime really acted as a bridge between us
- Focused on Cholesky factorization
 - Findings applicable to dense linear algebra in general
- Improved performance bounds
- Injected static knowledge of the application into runtime scheduler



Scheduling beyond state of the art

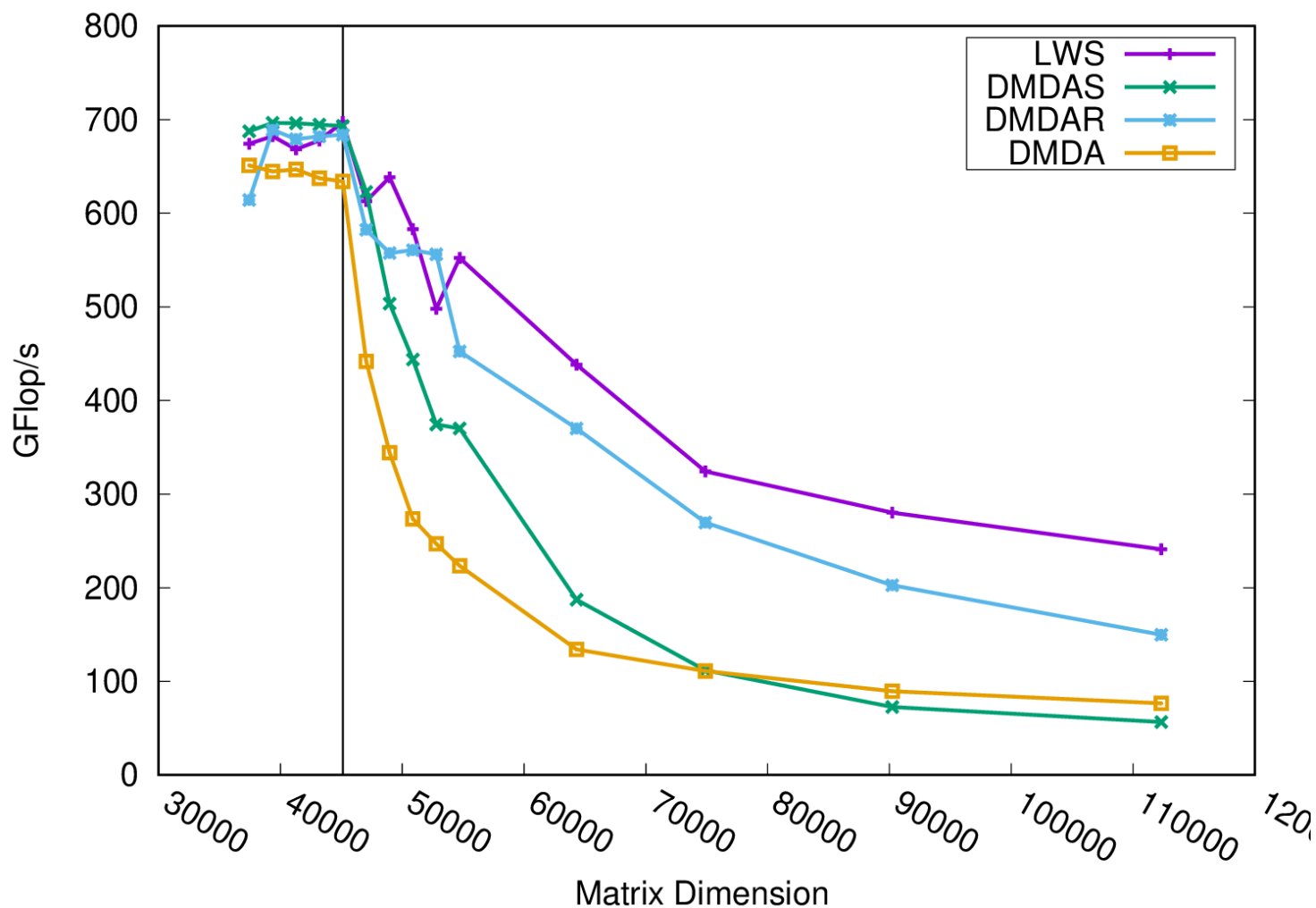
- Dense Cholesky factorization

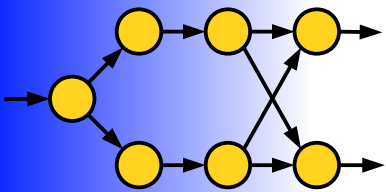




Scheduling beyond state of the art

- Dense Cholesky beyond memory size (out-of-core)





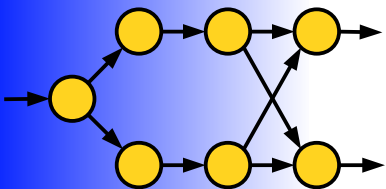
Scheduling beyond state of the art

Hi-BOX project with Airbus

- H-matrices
 - Hierarchically compressed blocks
 - Growing data
- e.g. 1 600 GB result on 256 GB system
- Spends half the time exchanging data with disk

Still need to find proper compromise between

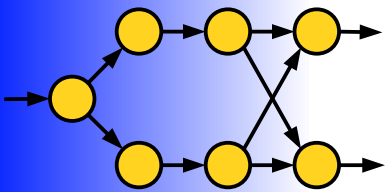
- critical path,
- acceleration,
- and data transfer



Getting theoreticians in?

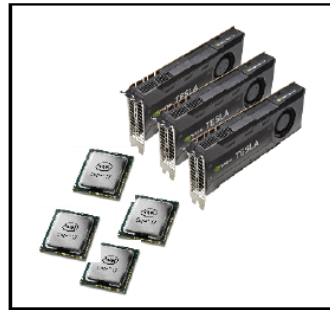
- Component-based schedulers should help
- Running actual application + runtime remains a pain
 - Getting access to target platforms
 - Installing software
 - And dependencies!

→ Simulation

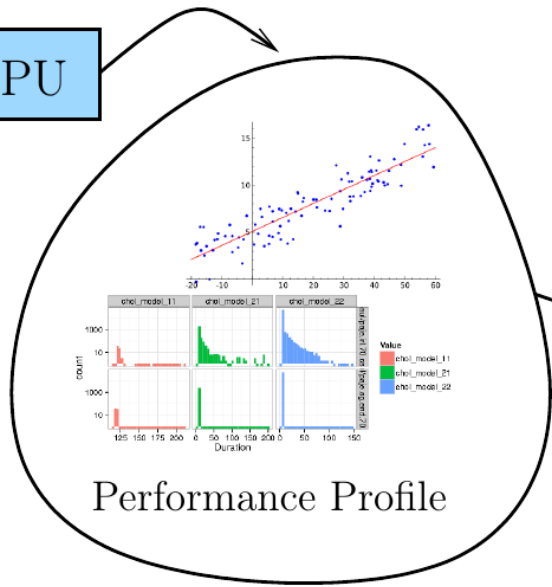


Simulation with SimGrid

Calibration



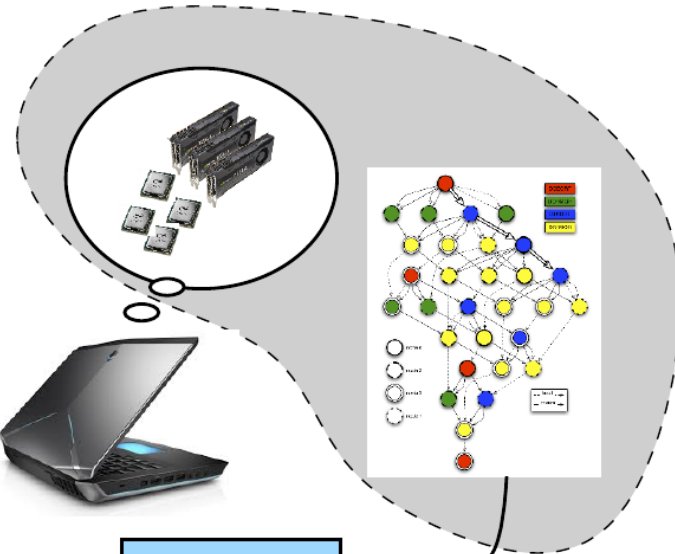
StarPU



Performance Profile

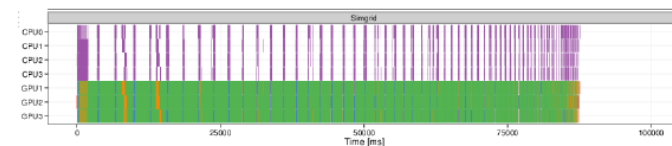
Run once!

Simulation



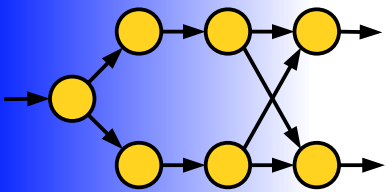
StarPU

SimGrid

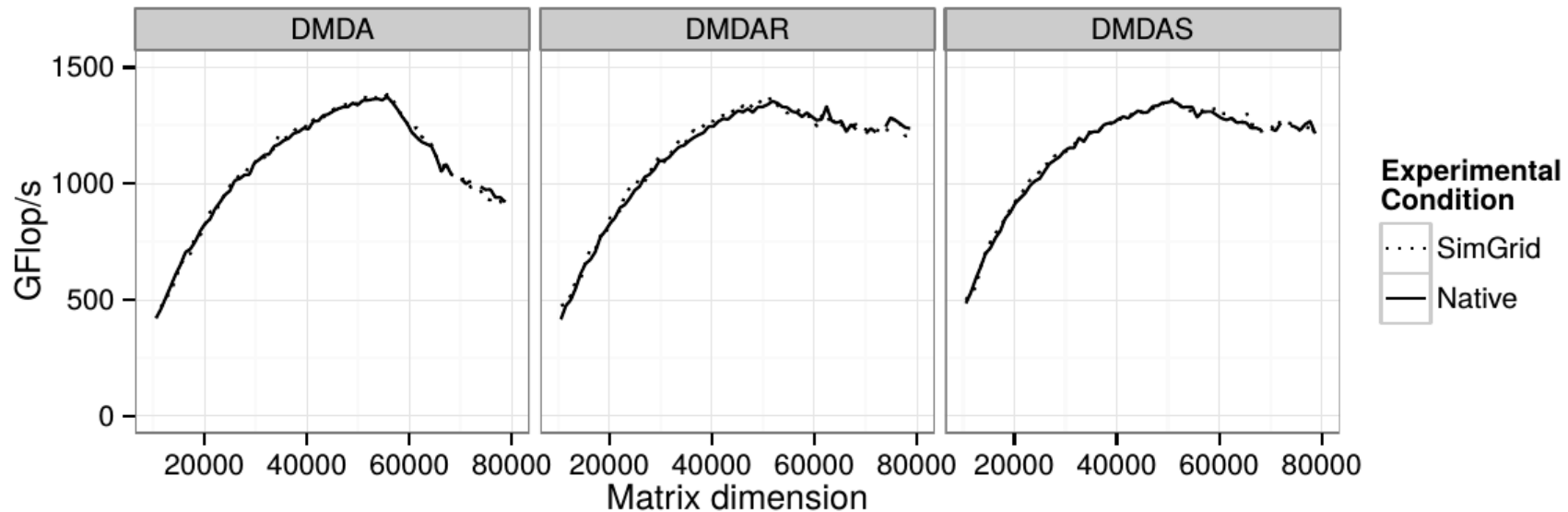


Quickly Simulate Many Times

With A. Legrand
and L. Stanislac



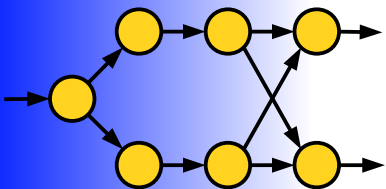
Simulation with SimGrid



Execute real application in simulation mode

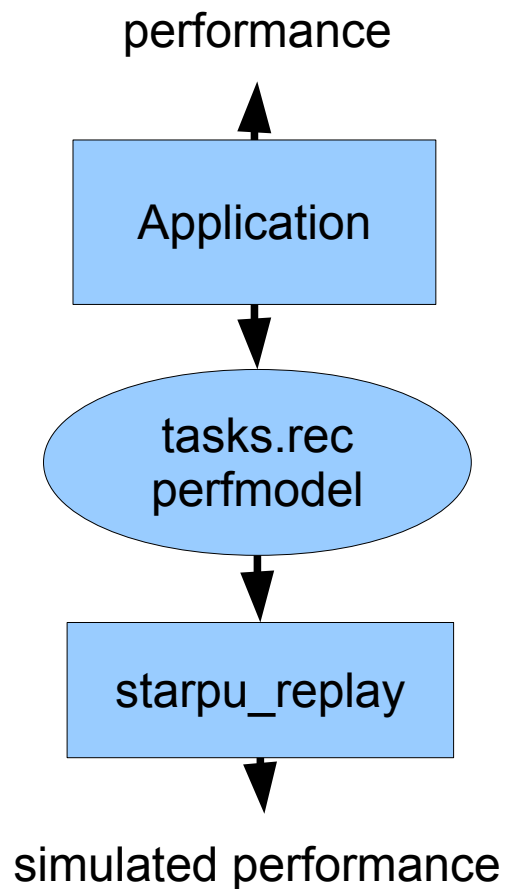
- Way faster execution time
- **Reproducible** experiments
- No need to run on target system
- Can change system architecture

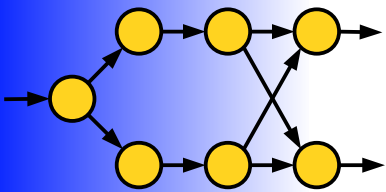
But still requires to build real application



Beyond Simulation with SimGrid

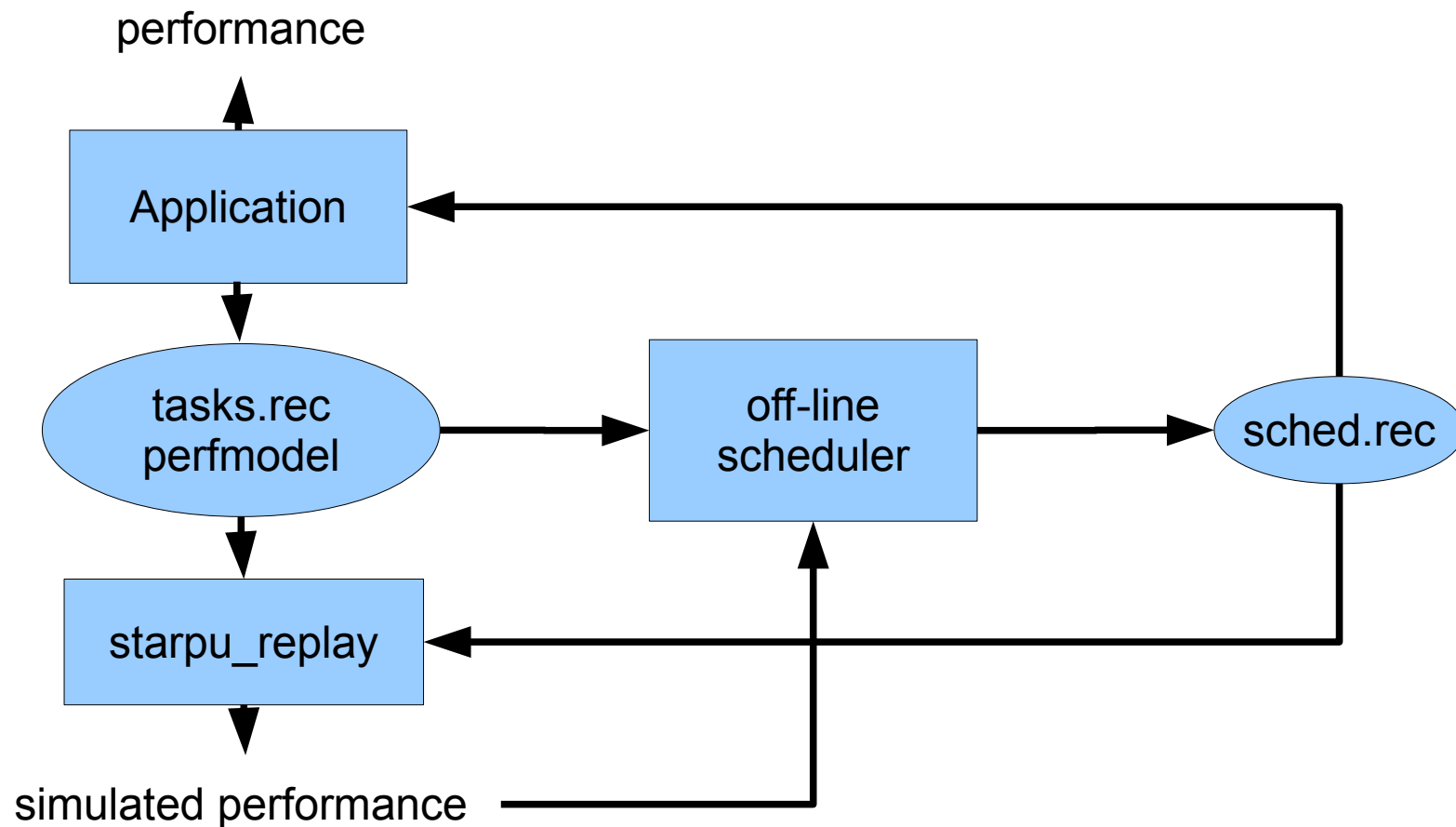
- Record application task graph
- Replay it with just `starpu_replay`

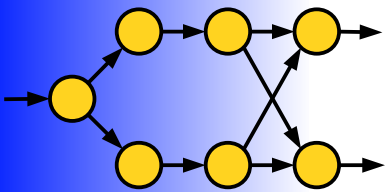




Beyond Simulation with SimGrid

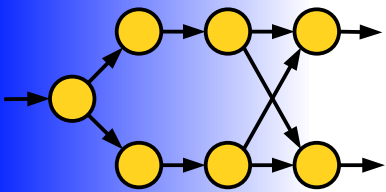
- Record application task graph
- Replay it, possibly with offline scheduling





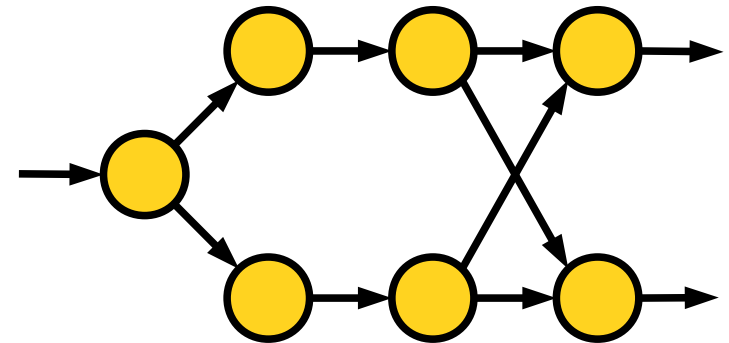
A task graph market?

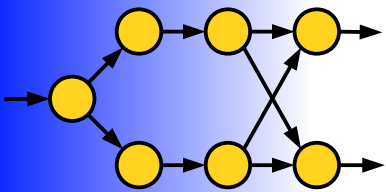
- Record application task graph
 - Varying application algorithms, dwarves, sizes
 - As many testcases for theoreticians
- Several levels of scheduling research
 - Just offline scheduling with tasks.rec
 - With high-level languages
 - Implement a real StarPU component
 - Run with starpu_replay
 - Care more about realworld conditions
 - Scheduler speed, still with starpu_replay
 - Run real applications in simulation
 - Run for real



Today's agenda

- Task graph scheduling
 - What is a runtime scheduler?
 - Beyond classical HEFT
 - Involving theoreticians
- Dividable tasks
- Distributed execution





How big should a task be?

Smaller task granularity

- Exposing parallelism
- Fine-grain load balancing

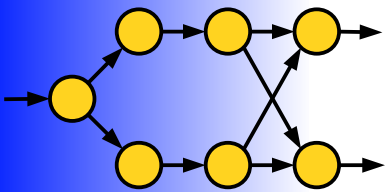
Large task granularity

- Needed by GPUs for efficiency

Making a compromise?

- Not very conclusive

→ Adaptative task size?

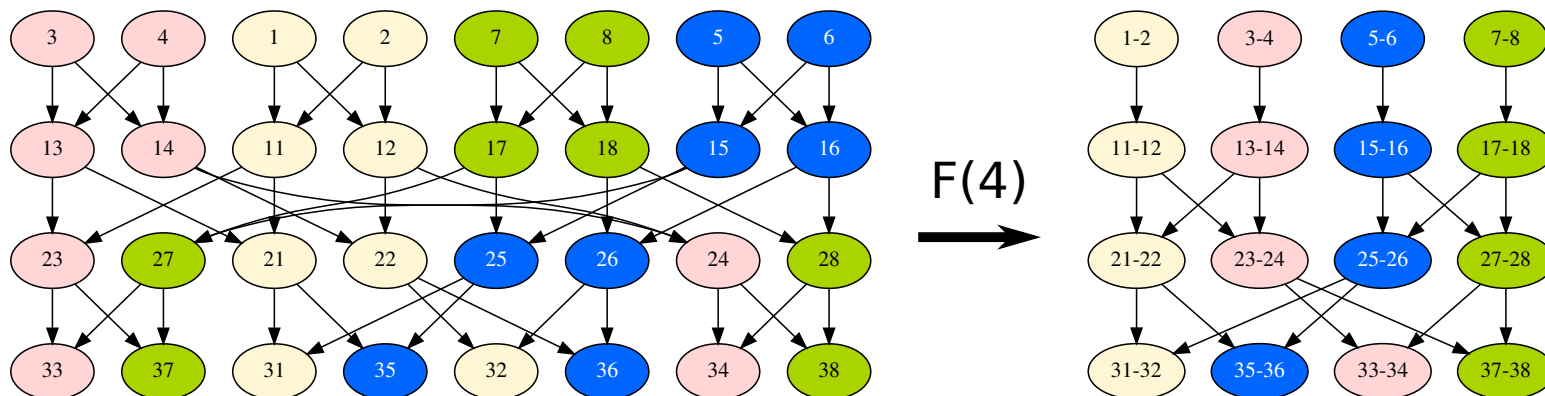
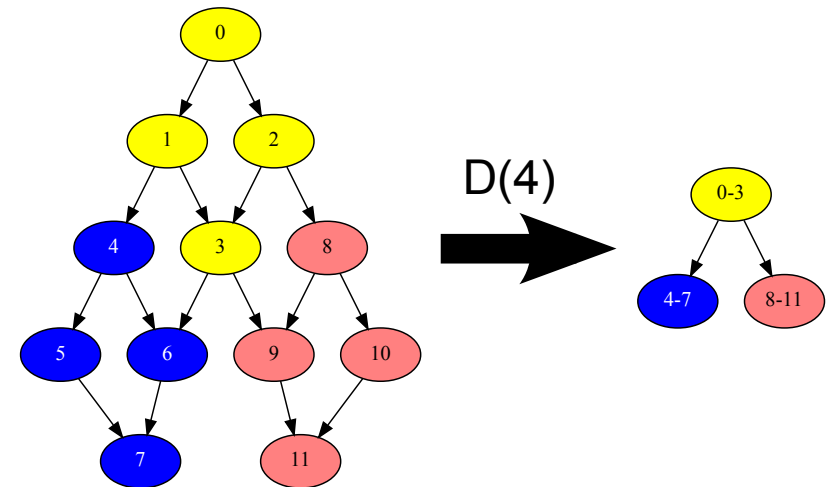


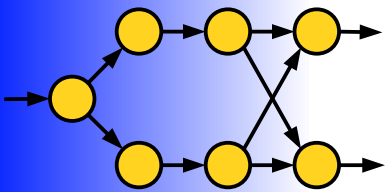
Gathering tasks

[RossignonPhD15]: Taggre

- Fine-grain task graph
- Gathering recipes
- Before submission to runtime

Allows to tinker with granularity without modifying application

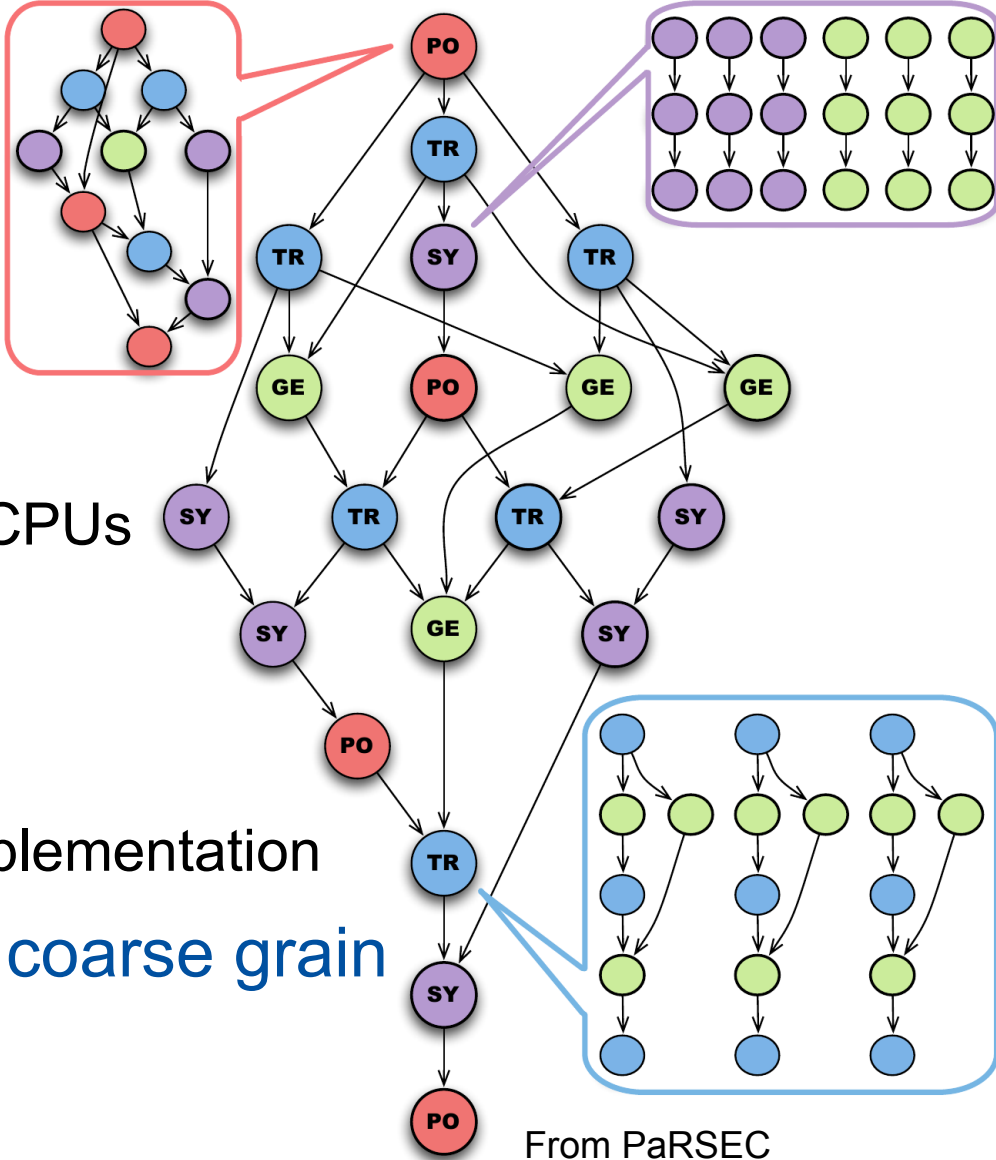




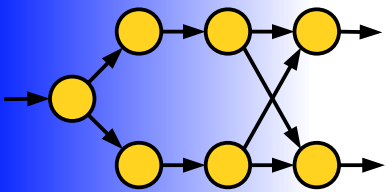
Dividing tasks

Recursive expression of graph

- Dividable tasks
- Dynamically adapt granularity
 - both big tasks for GPUs
 - and small tasks for numerous CPUs
- Similar to thread bubbles
 - Called so in current StarPU implementation
- Apply complex scheduling at coarse grain
 - For $O(n^2)$ / $O(n^3)$ algorithms...



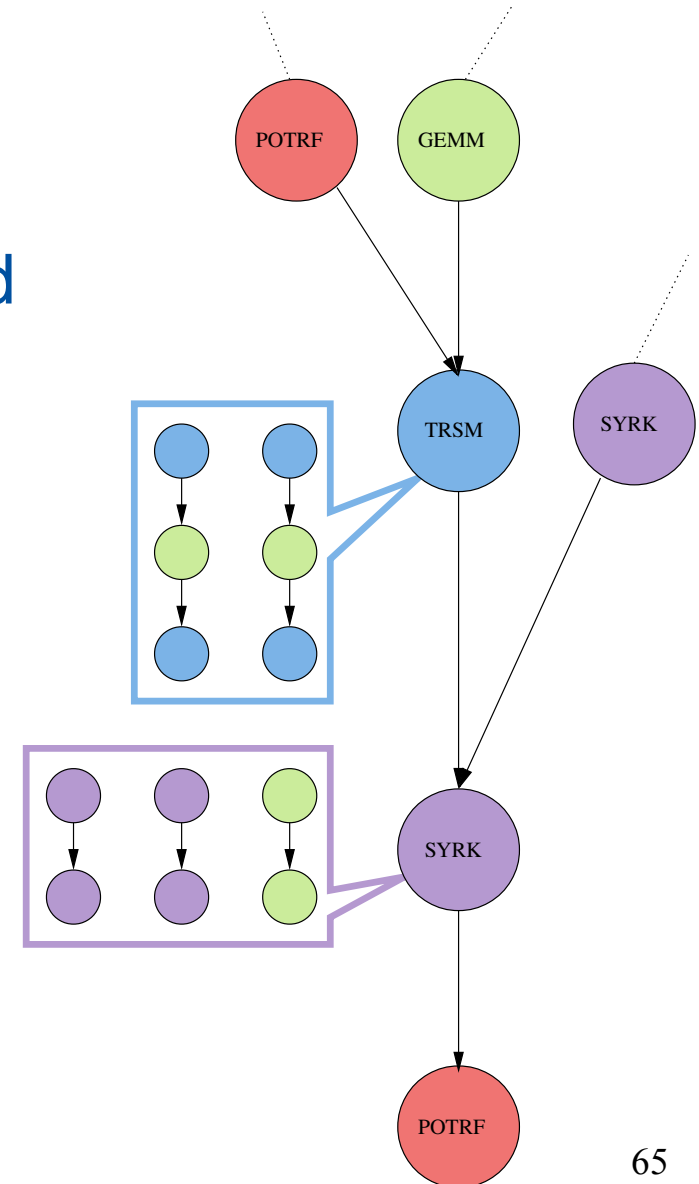
From PaRSEC

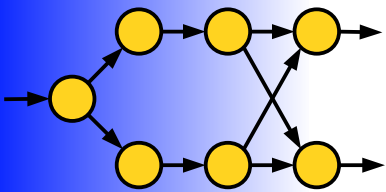


Dividing tasks

Synchronization concerns

- Fork-join parallelism
- Hindered by synchronization induced by task graph





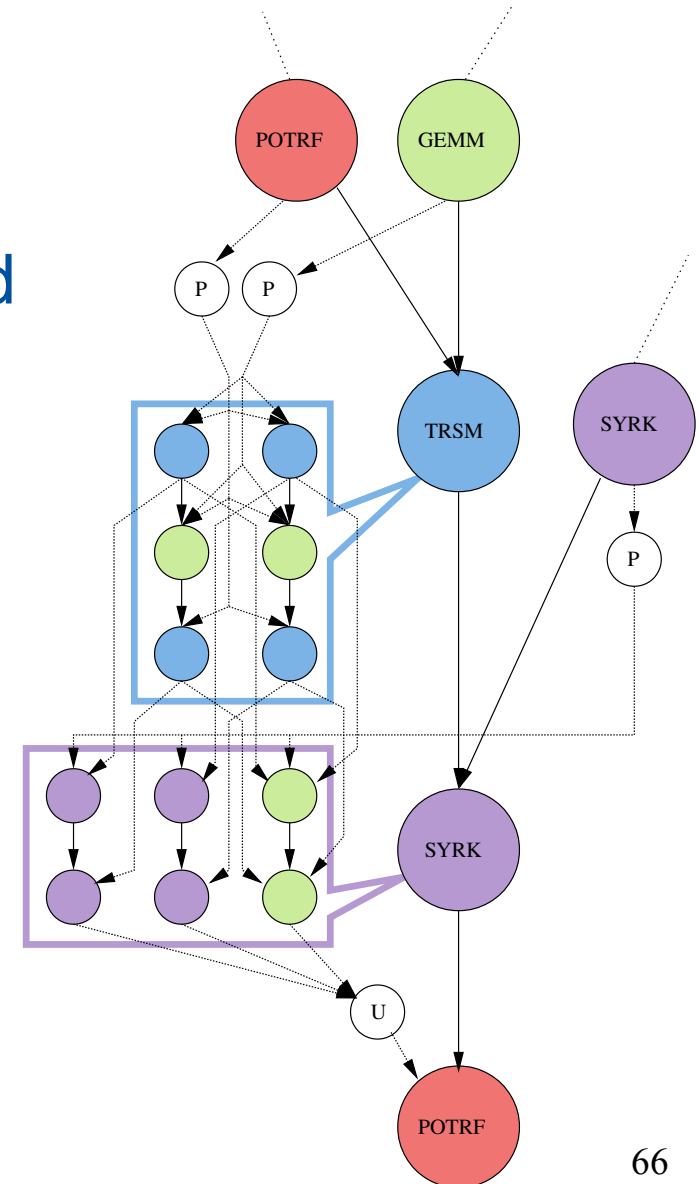
Dividing tasks

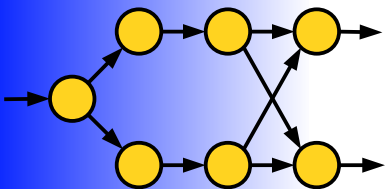
Synchronization concerns

- Fork-join parallelism
- Hindered by synchronization induced by task graph

Multi-level data coherency

- Synchronization pseudo-tasks
 - Only when needed
- Result is exactly as appropriate

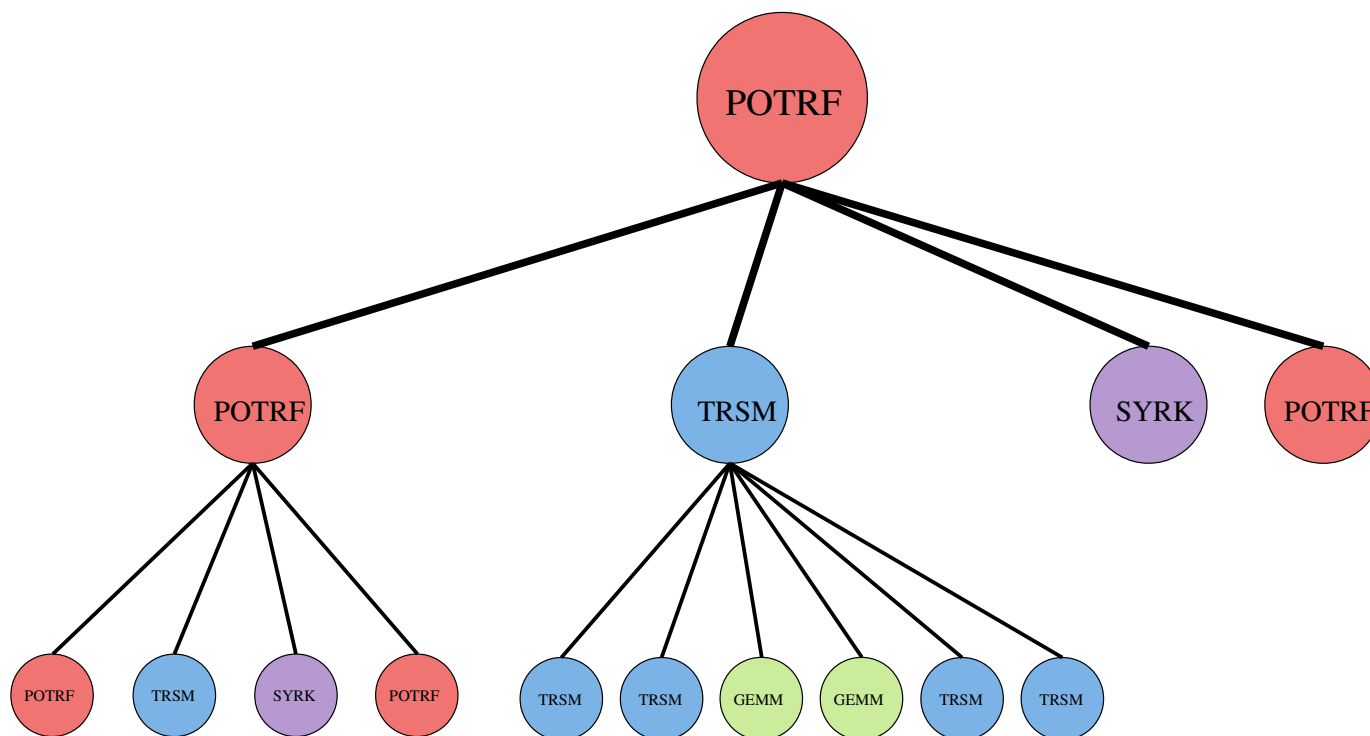




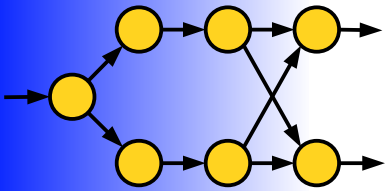
Dividing tasks

No extra synchronization

→ Can consider task graph subdivision as a tree



Dividing tasks



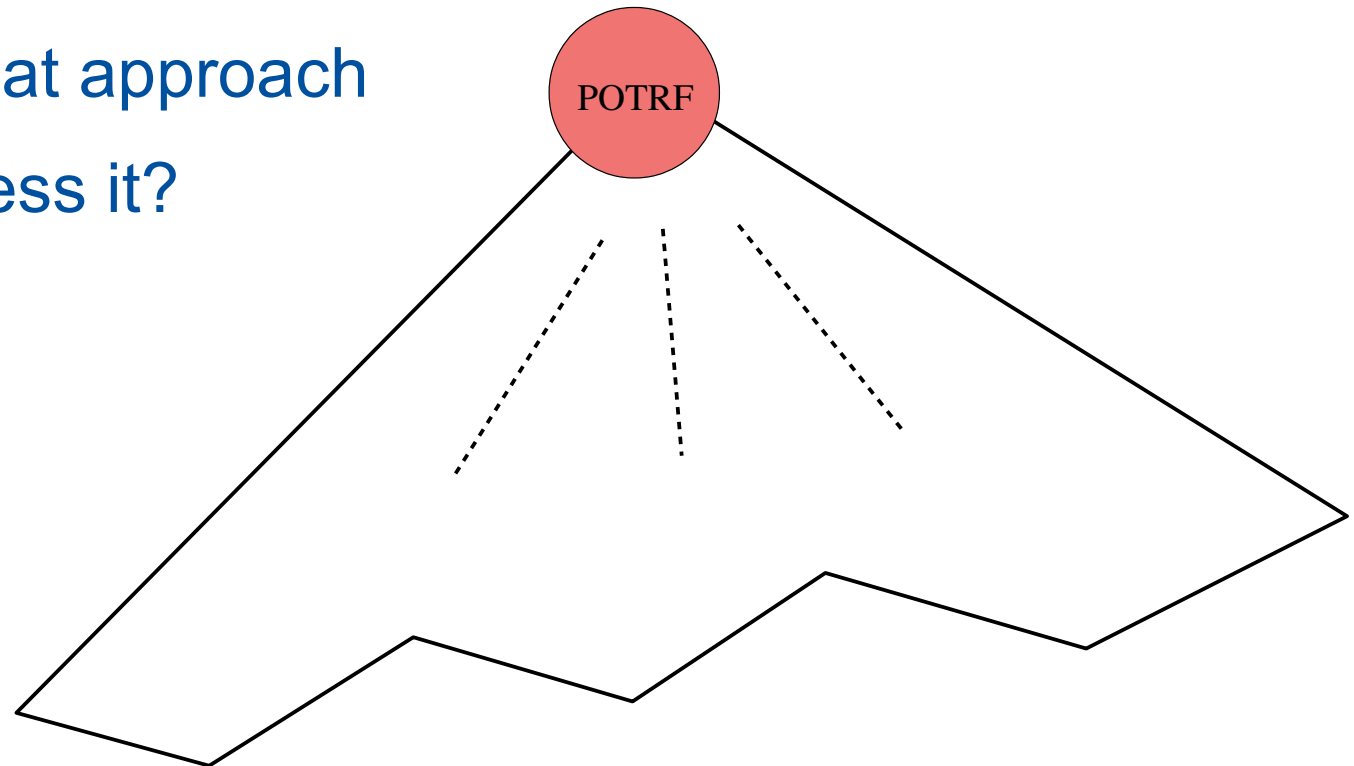
No extra synchronization

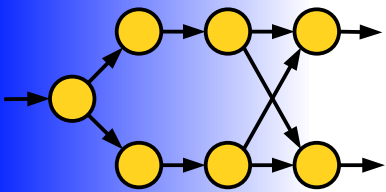
→ Can consider task graph subdivision as a tree

→ Decide at will where and when to stop recursing

Actually Airbus' hmat approach

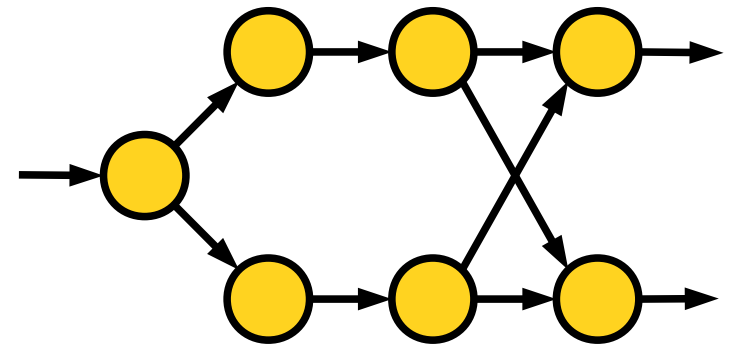
How to nicely express it?

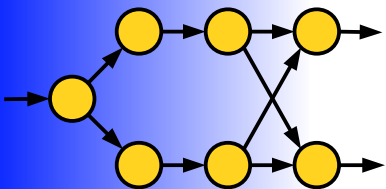




Today's agenda

- Task graph scheduling
 - What is a runtime scheduler?
 - Beyond classical HEFT
 - Involving theoreticians
- Dividable tasks
- Distributed execution

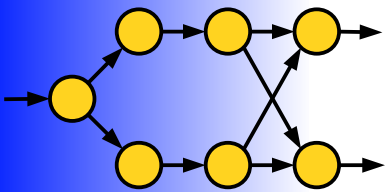




Distributed execution

Try to keep STF principle

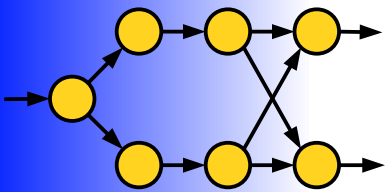
- Master-slave mode
 - Master unrolls whole task graph
 - Master schedules tasks between slaves
 - Just like scheduling between CPUs and GPUs
 - Limited scaling



Distributed execution

Try to keep STF principle

- ~~Master-slave mode~~
- Completely distributed mode
 - Application provides data mapping
 - Task mapping according to data mapping
 - Task run on node which owns data written to
 - All nodes unroll the whole task graph
 - Only keep tasks they have to execute
 - Automatically generate communications



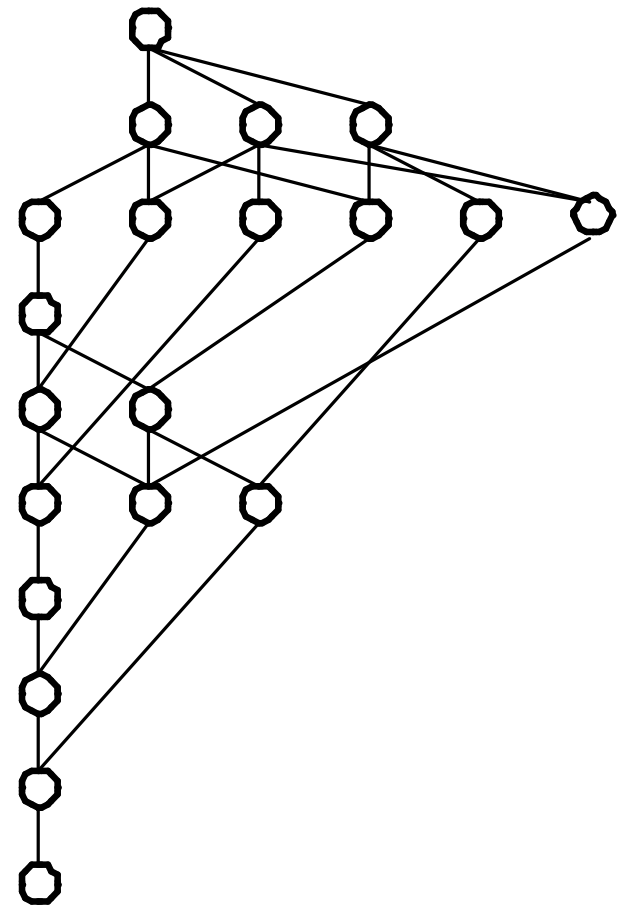
Distributed execution

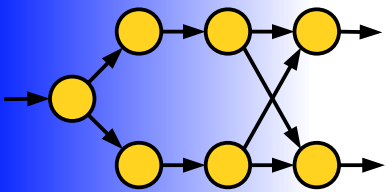
Fully distributed with STF

```

For (k = 0 .. tiles - 1) {
  POTRF(A[k,k])
  for (m = k+1 .. tiles - 1)
    TRSM(A[k,k], A[m,k])
  for (m = k+1 .. tiles - 1) {
    SYRK(A[m,k], A[m,m])
    for (n = m+1 .. tiles - 1)
      GEMM(A[m,k], A[n,k], A[n,m])
  }
}

```





Distributed execution

Fully distributed with STF

```
int get_rank(int m, int n) { return ((m%p)*q + n%q); }
```

```
For (m = 0 .. tiles - 1)
```

```
  For (n = m .. tiles - 1)
```

```
    set_rank(A[m,n], get_rank(m,n));
```

```
For (k = 0 .. tiles - 1) {
```

```
  POTRF(A[k,k])
```

```
  for (m = k+1 .. tiles - 1)
```

```
    TRSM(A[k,k], A[m,k])
```

```
  for (m = k+1 .. tiles - 1) {
```

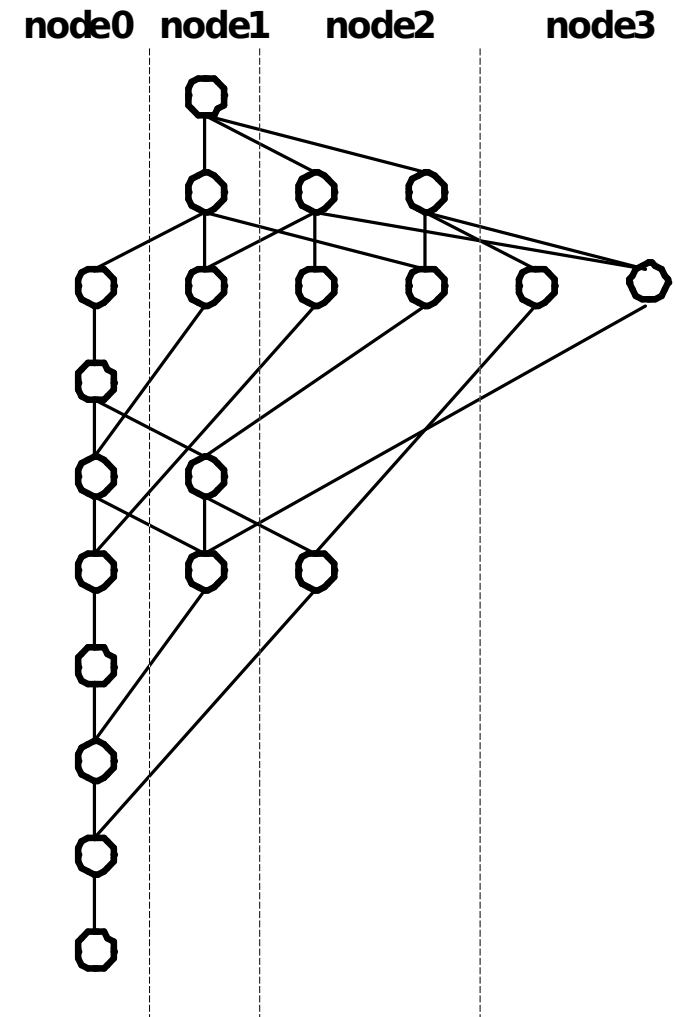
```
    SYRK(A[m,k], A[m,m])
```

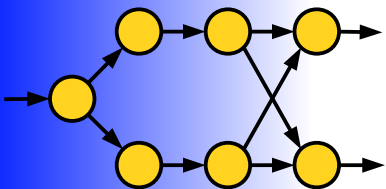
```
    for (n = m+1 .. tiles - 1)
```

```
      GEMM(A[m,k], A[n,k], A[n,m])
```

```
  }
```

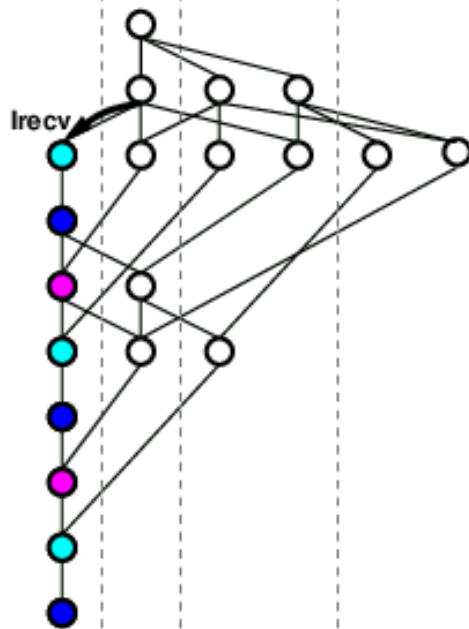
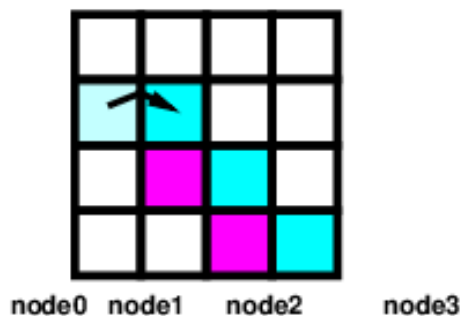
```
}
```



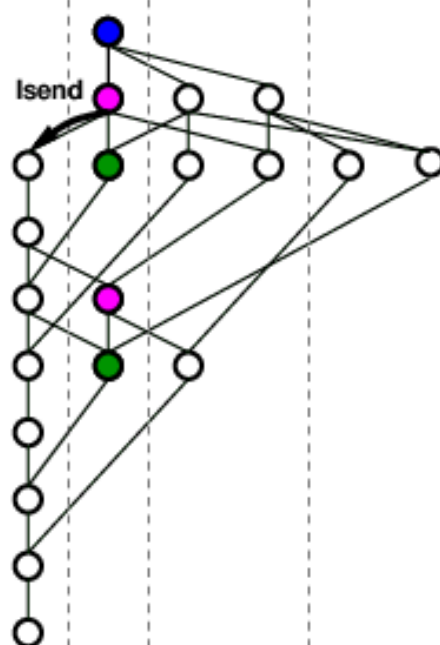
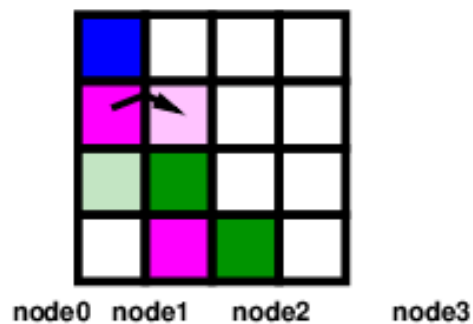


Distributed execution

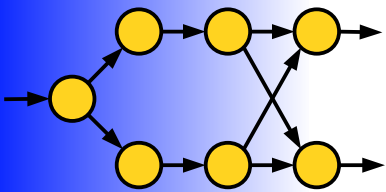
Fully distributed with STF



Node 0 execution



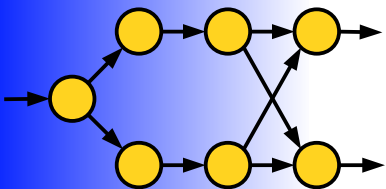
Node 1 execution



Distributed execution

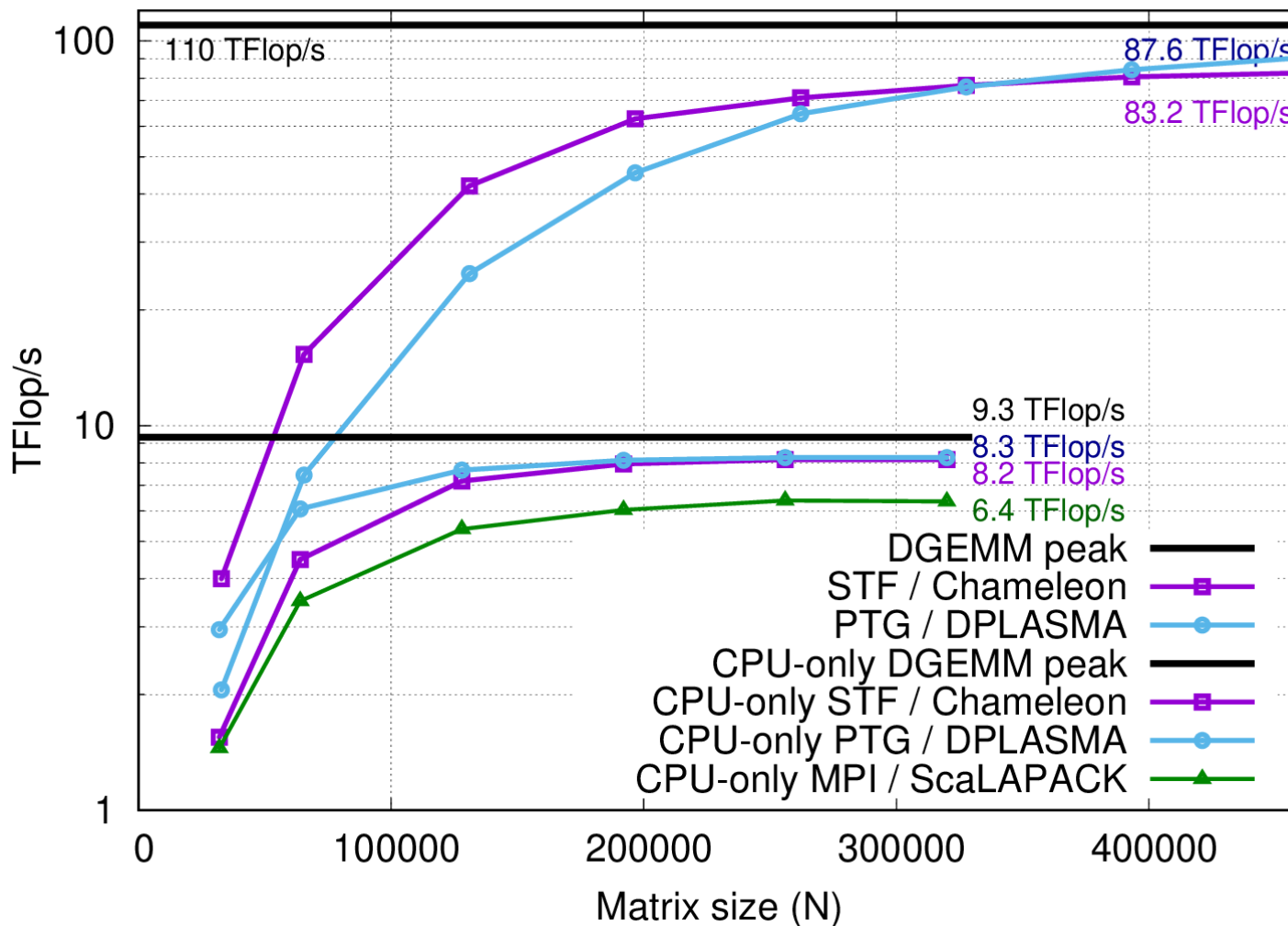
[SergentPhD16]

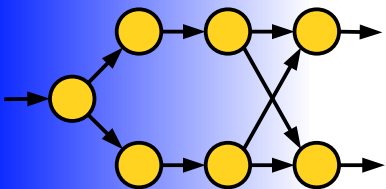
- Making it to scale
 - Caching values
 - Pruning task graph
- Require less memory
 - Throttling submission



Distributed execution

Result competitive with state of the art, over 144 nodes
(1152 CPU cores, 288 GPUs)

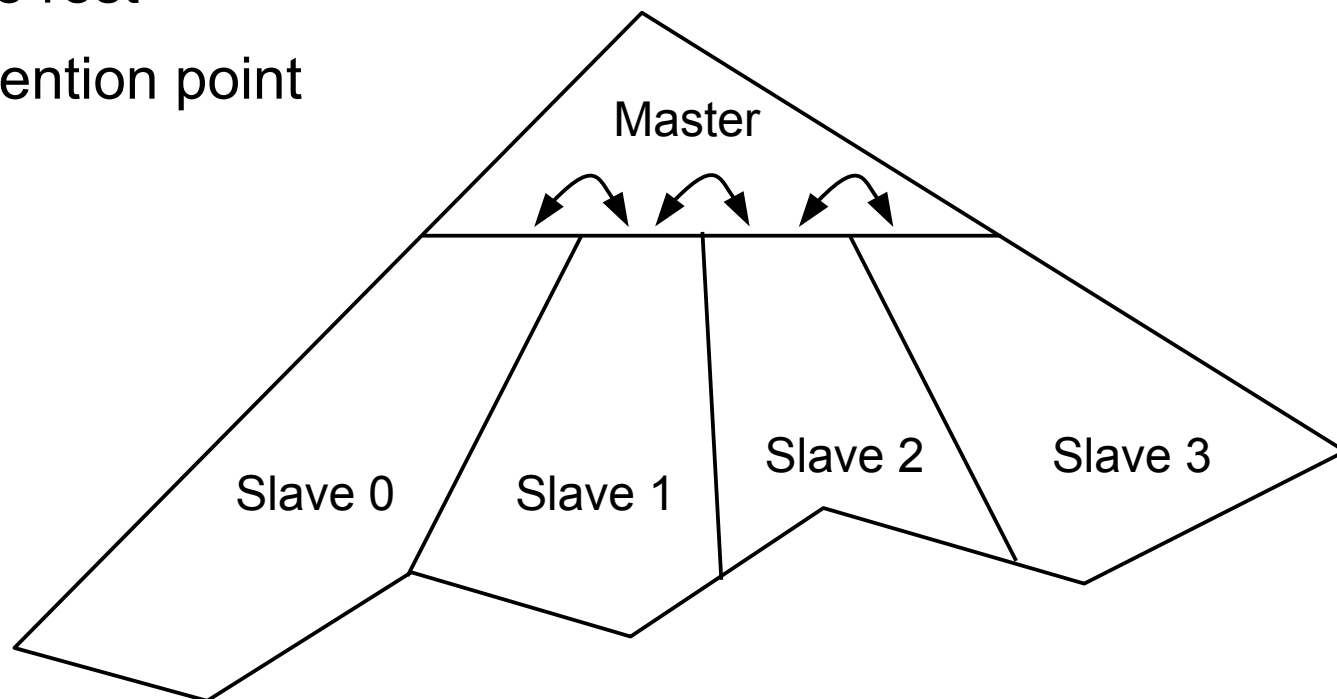


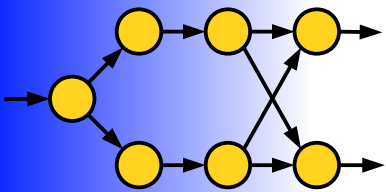


Scaling at large

Could leverage dividable tasks

- ClusterSs from BSC
 - Master unrolls higher recursion levels, schedules result
 - Slaves unroll the rest
 - Master still contention point



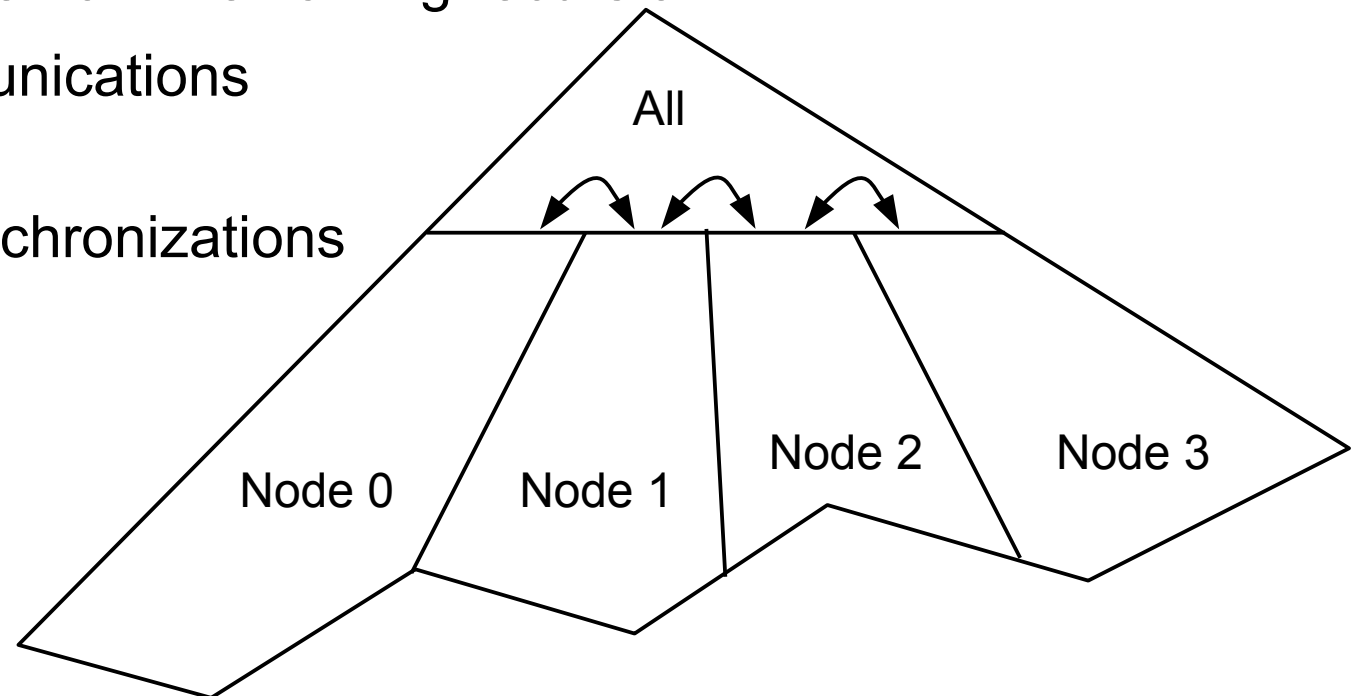


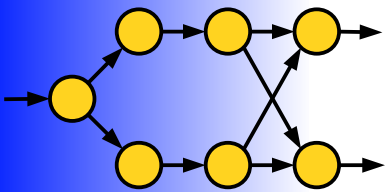
Scaling at large

Could leverage dividable tasks

- **DuctTeip from University of Uppsala**

- All nodes unroll higher recursion levels, determine task mapping
- Nodes unroll their own remaining recursion
- Network communications quite coarse
→ spurious synchronizations



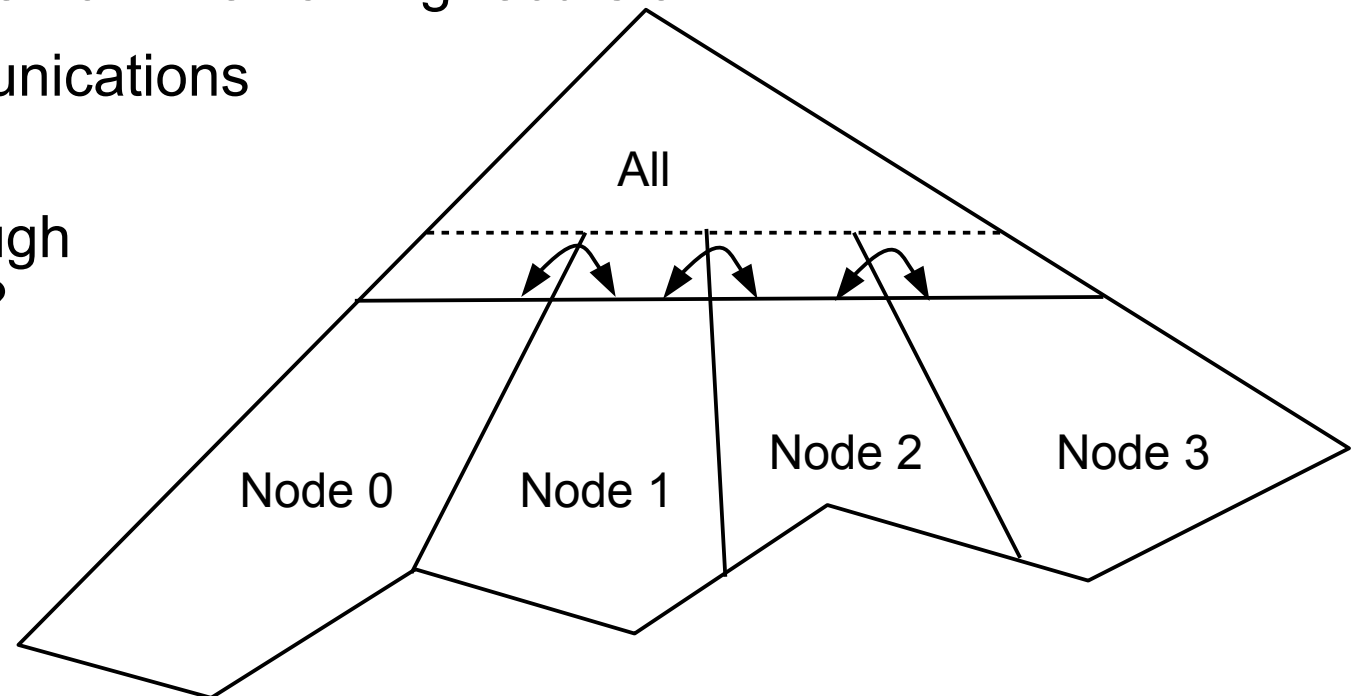


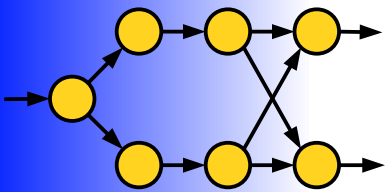
Scaling at large

Could leverage dividable tasks

- Ideally?

- All nodes unroll higher recursion levels, plus some margin
- Nodes unroll their own remaining recursion
- Network communications at finer grain
- Fine-grain enough vs overhead?

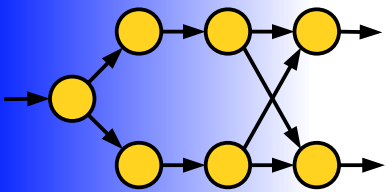




Conclusion

Heterogeneous applications on heterogeneous platforms

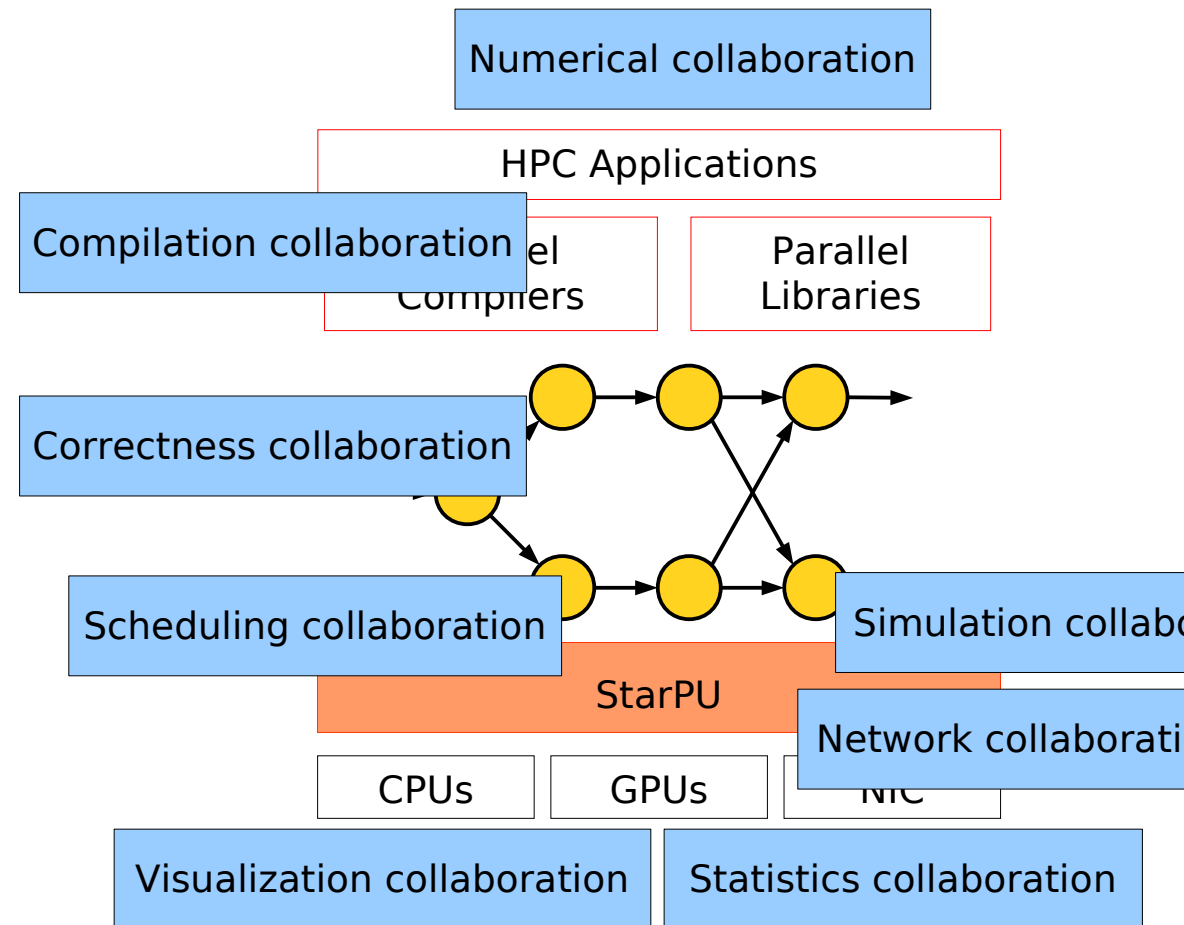
- Expressed with sequential-looking code (STF)
- Interesting scheduling problem
- Built bridge between scheduling theory and applications
- Dividable tasks to better address heterogeneity
- Fully distributed execution competitive



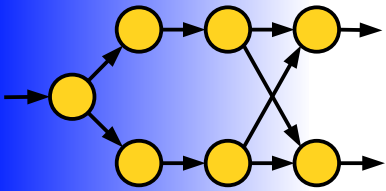
Conclusion

A lot of expertise gathered around StarPU

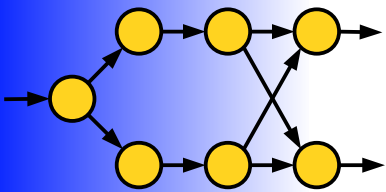
- Thanks to task graph
- Expressed with STF
- StarPU as a bridge between theory and applications



Perspectives



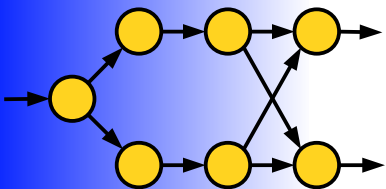
- Programming interface (OpenMP?, dividable tasks, BigData)
- Scheduling
 - Tasks
 - Memory ([ArrasPhD15])
 - Moldable tasks
- Composition (runtimes, programming paradigms)
- Leveraging compilers
- Visualization
- Model checking
- Fault tolerance of distributed execution
 - Checkpoint / restart + distributed load balancing [LionPhD]



Thanks!

[AugonnetPhD11] [ArrasPhD15] [RossignonPhD15]
[SergentPhD16] [KumarPhD17] [LionPhD]

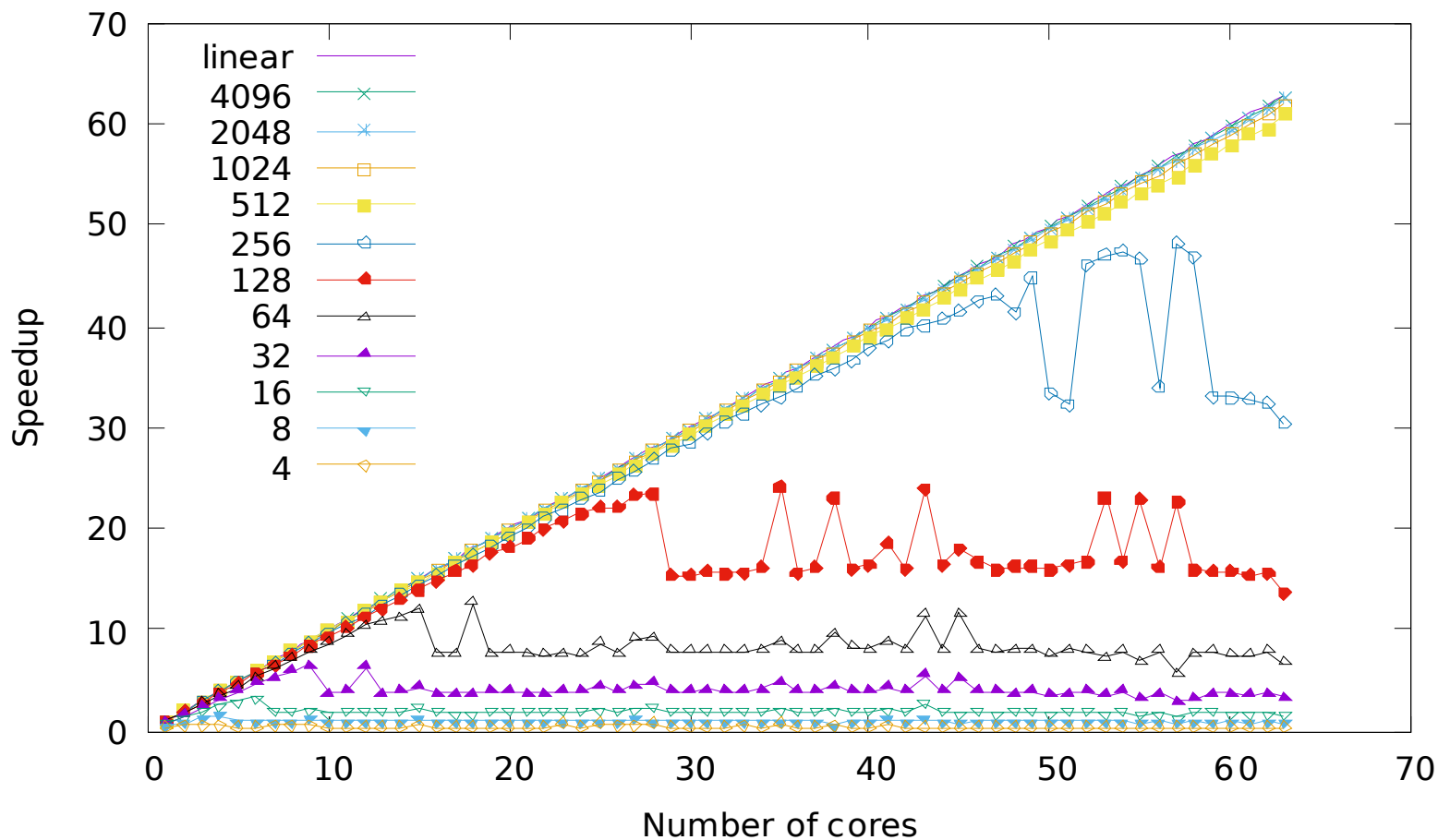
- ANR ProHMPT'09, MEDIAGPU'10, FP3C'10, SONGS'12, SOLHAR'13
- IPL HAC-SPECIS'16, HPC-BigData'18
- EU PEPPER'10, EXA2PRO'18
- Rapid Hi-BOX'13
- MORSE associate-team
- Used by AL4SAN, Chameleon, ExaGeoStat, FLUSEPA, HiCMA, hmat, KSVD, MAGMA, MaPHYs, MOAO, PaStiX, QDWH, qr_mumps, ScalFMM, SCHNAPS, SignalPU, SkePU, STARS-H

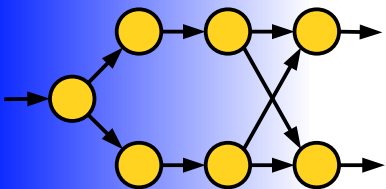


How big should a task be?

Lower bound due to runtime overhead

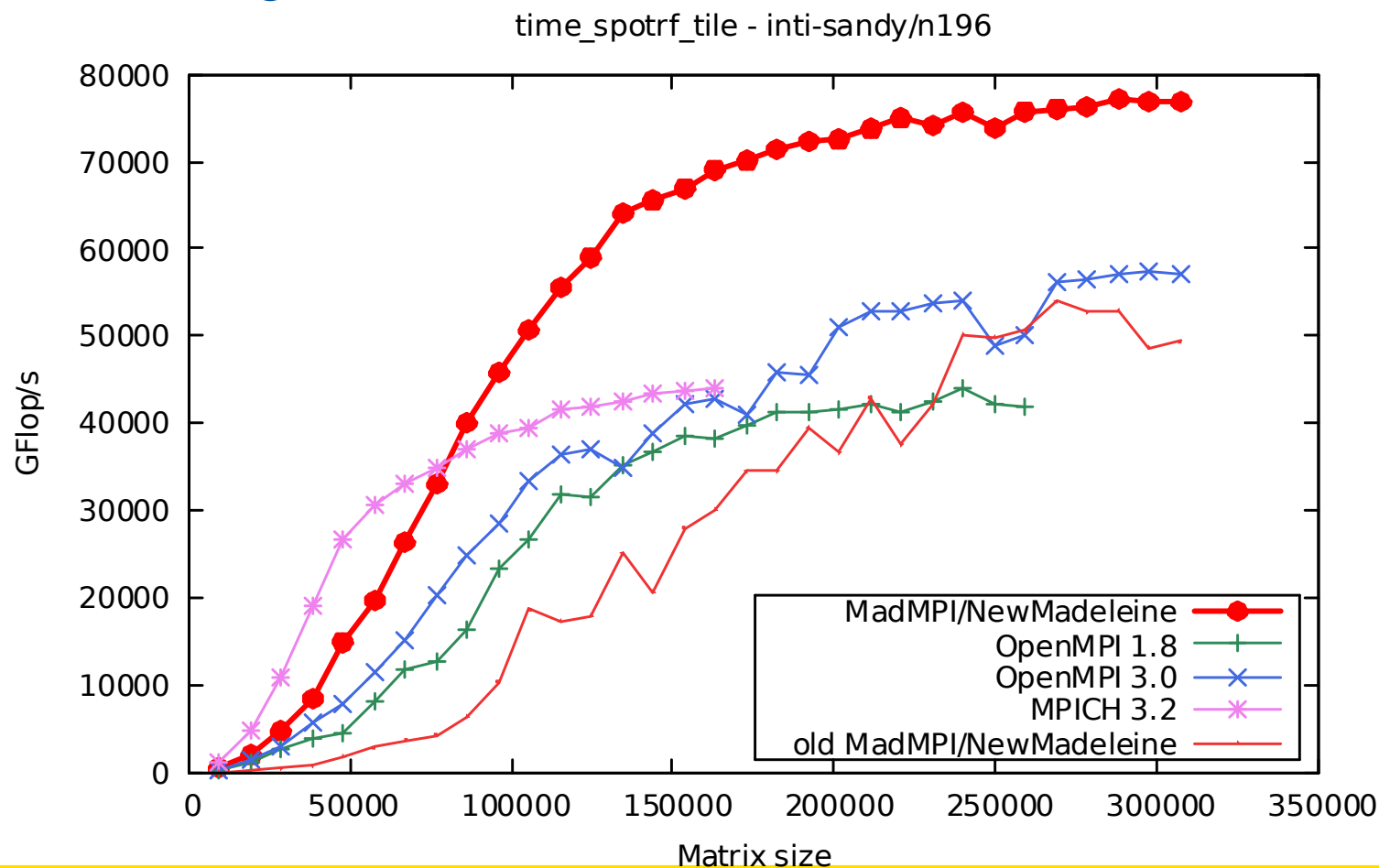
- Proposed by Martin Tillenius

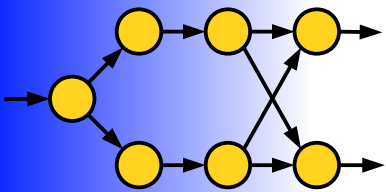




Issues with MPI implementations

- Classical MPI implementations not used to irregular communications
- Collaborating with A. Denis





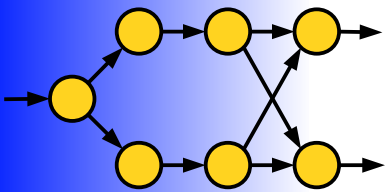
Scaling at large

All nodes unrolling all loops expensive

- Task graph pruning

```
#define GEMM(A, B, C) \  
if (get_rank(A) == self || get_rank(B) == self || get_rank(C) == self) \  
GEMM(A, B, C);
```

```
For (k = 0 .. tiles - 1) {  
    POTRF(A[k,k])  
    for (m = k+1 .. tiles - 1)  
        TRSM(A[k,k], A[m,k])  
    for (m = k+1 .. tiles - 1) {  
        SYRK(A[m,k], A[m,m])  
        for (n = m+1 .. tiles - 1)  
            GEMM(A[m,k], A[n,k], A[n,m])  
    }  
}
```



Scaling at large

All nodes unrolling all loops expensive

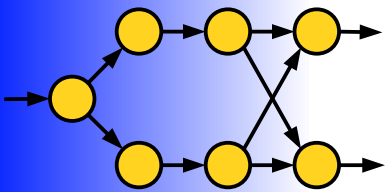
- Could use source-to-source compilation to rework loop nest
 - Polyhedral analysis, to get

```

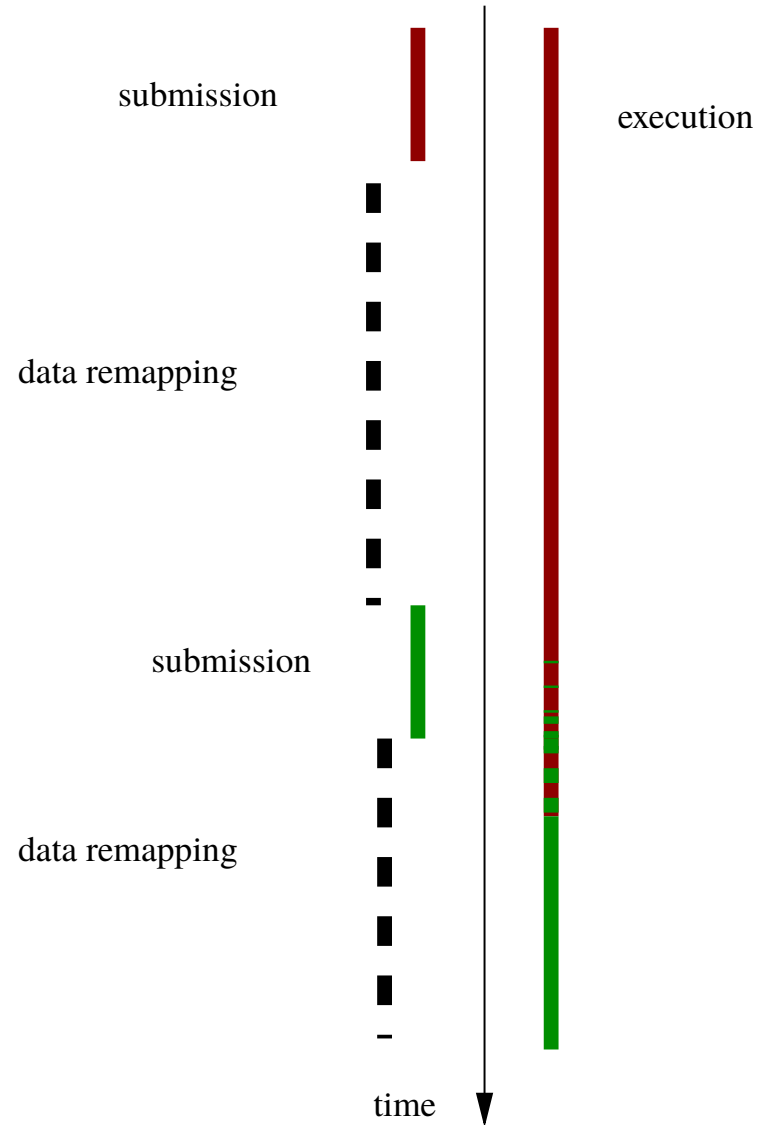
For (k = 0 .. tiles - 1) {
  POTRF(A[k,k])
  for (m = k+1 .. tiles - 1)
    TRSM(A[k,k], A[m,k])
  for (m = k+1 .. tiles - 1) {
    SYRK(A[m,k], A[m,m])
    for (n = m+1+xxx .. tiles - 1 step Q)
      GEMM(A[m,k], A[n,k], A[n,m])
  }
}

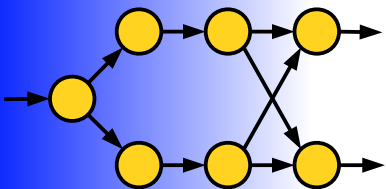
```

- Reduce complexity from $O(n^3)$ to $O(n^3/Q)$



Dynamic data remapping





Checkpoint / restart

